Englisch · Gerits · Thrun
Hartwig · Löffelmann

64 Tips & Tricks

BAND 1

Der Bestseller mit
100.000
Exemplaren
Gesamtauflage

Eine Fundgrube für den Commodore 64 Anwender

EIN DATA BECKER BUCH

Wichtiger Hinweis: Da uns aus drucktechnischen Gründen das Zeichen '↑' nicht zur Verfügung steht, benutzen wir in diesem Buch statt dessen das Zeichen '^'. Wir bitten Sie, dies beim Eingeben der Programme zu beachten.

Englisch · Gerits · Thrun
Hartwig · Löffelmann

64 Tips & Tricks

Eine Fundgrube für den Commodore 64 Anwender

EIN DATA BECKER BUCH

ISBN 3-89011-001-0

5. erweiterte und überarbeitete Auflage

Copyright © 1986 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Inhaltsverzeichnis:

1. Kap. - Effektive BASIC-Programmierung 13

1.1 Optimierung von Arrays 15
1.2 Realisierung von ESP-Programmen 16
1.3 Vor-Definition von Variablen 18
1.4 Ausschalten des Bildschirms 20
1.5 C-Print-Befehl 21
1.6 Strukturen 22
1.7 Space-Tracer 23
1.8 Makros mit dem V2.0-BASIC 24
1.8.1 Zentrierter Ausdruck 25
1.8.2 Logarithmus 26
1.8.3 Modulo 27
1.8.4 Deg-Rad Umwandlung 28
1.8.5 Maxi-Division auf alle Stellen 29
1.8.6 Fakultät 30
1.8.7 Def Fn, das unbekannte Wesen 30
1.9 Anti-Error-Routine 33
1.10 Die 3 Uhren des C-64 36
1.11 Adressierung der internen TOD-Clock 38
1.12 Statement-Flasher 43
1.13 Revers 44
1.14 Var-Tracer 44
1.15 Micro-Del-Routine 46
1.16 Mikro-Auto-Line-Routine 49
1.17 Aufspüren von 'Out of Stack Memory Errors' 50
1.18 Cursorpositionierung 53
1.19 Modifizierter Print-Befehl 55
1.20 Komfortable Input-Routinen in BASIC und Assembler 59
1.21 Errechnen beliebiger Formeln 78
1.22 Datazeilengenerator 84
1.23 Bildschirmmaskengenerator 85
1.24 Rückkehr ins Basic-Programm nach List 93
1.25 Berechnete Zeilennummern bei Gosub, Goto und Restore 96
1.26 Der Kassettenpuffer als Programmspeicher 99
1.27 Bubblesort in Assembler 101

2. Kap. - Grafik für Fortgeschrittene

2.1 Die Textgrafik 107
2.2 Grafikanimation 109
2.2.1 Zweifarbiger Bildschirmrand 115
2.2.2 Multicolor-Border-Farben 115
2.2.3 Kombinierte Border- und Paperfarbe 116
2.3 Extended Colour TEC-Modus 120
2.3.1 TEC-Referenzliste 125
2.4 Hochauflösende Gafik 127
2.4.1 Punktberechnung in der Hi-Res-Grafik 128
2.4.2 Ein kleines Grafikpaket 134
2.4.3 3D-Grafik 137
2.5 Zeichensatzmodifikation 144
2.5.1 Verschieben des Zeichensatzes 145
2.5.2 Verändern des Zeichensatzes 147
2.6 Die Sprites 155
2.6.1 Kollision von Sprites 156
2.6.2 Ein komfortabler Sprite-Editor 157
2.6.3 Erklärung des Sprite-Editors 169
2.7 Rasterzeileninterupt - der geteilte Bildschirm 171

3. Kap. - Soundprogrammierung - 189

3.1 Soundanimation 191
3.1.1 Datasette-Sound 191
3.1.2 Sound-Filter 192
3.1.3 Sound-Verbesserung 192
3.1.4 Sound-Dämmung 193
3.1.5 Soundroutinen 193
3.2 Synthesizer in Stereo 199
3.3 Sounderweiterung für den C-64 203

4.Kap. - Benutzerschnittstellen - 219

4.1 Maussimulation mit dem Joystick 221

4.2 Der Joystick als komfortable Cursorsteuerung 225
4.3 GEM-Simulation auf dem C-64 228
4.4 Ausgabe deutscher Fehlermeldungen 232
4.5 Tastaturbelegung und ihre Änderung 237
4.6 Die Belegung der Funktionstasten 241

5.Kap. - Peripherie - 249

5.1 Ein kleiner Kopierschutz 251
5.2 Hardcopy auf Epson-Druckern 256
5.3 Schreibschutzdetektor 260
5.4 Disk-Versiegler 261
5.5 Disk-Entsiegler 262
5.6 Programme als Usr-/Seq-Files 264
5.7 Floppy-Reset 265
5.8 Close all 266
5.9 Komma-Files 268
5.10 Scratch-File-Recover 269
5.11 Directory-Kill 270
5.12 Selektiver Directory-Schutz 270
5.13 Directory-Steuercodes 273
5.14 Sondereinträge 274
5.15 Tape Speed-up 276

6. Kap. - Befehlserweiterungen - 279

6.1 Der erweiterte SYS-Befehl 281
6.2 Keine negativen Zahlen bei der Fre-Funktion 289
6.3 Die Erweiterung des Mid$-Befehls 291
6.4 Die Instr- und String$-Funktion 297
6.5 Automatische Zeilennumerierung 304
6.6 Die Benutzung von Token 307

7. Kap. - Schnittstellen - 317

7.1 Der User-Port 319
7.2 Registerbeschreibung des 6526 319

7.3 Eine sinnvolle Anwendung des User-Ports: Schnittstelle für einen Centronic-Drucker 323
7.4 Datenübertragung von und zu anderen Rechnern 330

8. Kap. - Kurz-Tips & Tricks - 337

8.1 Directory ohne Programmverlust 339
8.2 Ausschalten von RUN-STOP/RESTORE 340
8.3 Der Basiceditor als Textverarbeitung 340
8.4 Ein funktionierendes RE-NEW 341
8.5 Unterdrücken von "FORMULAR TOO COMPLEX" 341
8.6 Verkürzen der Floppyzugriffszeit 342
8.7 Finden von Datazeilen 342
8.8 Mehr Struktur in BASIC-Programmen 342
8.9 Input ohne Fragezeichen 343
8.10 Einfache BASIC-Hardcopy 343
8.11 Beenden eines Programms ohne "READY." 344
8.12 Prinzip eines Autostarts von Diskette 344
8.13 Programmierung der Modulstartes 345
8.14 Wenn sich der Computer weigert zu rechnen 345
8.15 Verhindern des List-Befehls 346
8.16 Vernünftiges Laden von Assemblerprogrammen 346
8.17 Das Beschleunigen von Programmen 346
8.18 Steuerung der Datasette 347
8.19 Erzwingen von Fehlermeldungen 347
8.20 Den Cursor frisieren 348
8.21 Kopie der Video-Controller 348
8.22 Ein simpler Kopierschutz 349
8.23 Scroll-Down in BASIC 349
8.24 Routine zum Kopieren des ROMs ins RAM 350
8.25 Ermitteln der aktuellen Fileparameter 350
8.26 Das Anhängen von BASIC-Programmen 351
8.27 GET# mit fünfacher Geschwindigkeit 351
8.28 Ready-Modifikation 352
8.29 INFormation 355
8.30 Unerlaubte Variablen 356
8.31 End-Reset 357
8.32 Funktionstasten-Splitting 357
8.33 Time-Delay-Loops 360

Anhang 361

1 Token-Referenzliste 363
2 ASCII-Tabelle 366
3 Kompletter Zeichensatz des C64 370
4 Umrechnungstabelle: - dezimal - hexadezimal - binär .. 376
5 Sprite-Entwurfsblatt 381
6 Expansion Port 382
7 User Port 383
8 VIC-Chip 384
9 SID-Chip 386
10 CIA-Chip 387
11 Kommentiertes Assemblerlisting zum Spriteeditor 388
12 Register 410

Kapitel 1

Effektive BASIC-Programmierung

1.1 Optimierung von Arrays

Eine der großen Schwächen der beim C-64 vorhandenen BASIC-Befehle stellt die DIM()-Anweisung dar.

Sind Variablenfelder in einem Programm erst einmal gesetzt, so können Sie nicht mehr verändert werden. Ein erneuter DIM-Befehl für ein bereits festgelegtes Array führt zu dem frustrierenden

```
?REDIM'D ARRAY ERROR
```

Die einzige Änderungsmöglickkeit von Feldern besteht darin, zunächst über CLR alle Variablen zu löschen und dann die neue Array-Dimensionierung durchzuführen. Der gravierende Nachteil hierbei ist jedoch, daß auch ALLE anderen Variablen wieder auf null gesetzt werden.

Fast immer werden im Programmablauf die alten Variablenwerte noch benötigt.

Man kann sich hier jedoch schnell helfen und einfach den ARRAY-Pointer zurücksetzen und somit nur die Variablenfelder, nicht aber die übrigen Variablen löschen.

Fügen Sie dazu die folgende Zeile an der entsprechenden Stelle in Ihr Programm ein:

```
POKE 49,PEEK(47)
POKE 50,PEEK(48)
```

Die Adressen 49/50 beinhalten dabei die Low- und High-Bytes des Arrayendes, 47/48 die des Feldanfangs.

Nach der Ausführung dieser beiden Befehle können alte Arrays, ohne einen REDIM'D ARRAY ERROR hervorzurufen wieder beliebig neu gesetzt werden, wobei alle herkömmlichen Variablen erhalten bleiben.

1.2 Realisierung von ESP-Programmen

ESP-Programme sind Programme, die scheinbar nur ihrem Autor, also Ihnen gehorchen, nicht aber beispielsweise einem Fremdanwender. Das Programm "kennt" Sie und verweigert jegliche Kooperation mit anderen Personen, die versuchen, damit zu arbeiten.

Es gibt vielfältige Möglichkeiten, ein derartiges Programm zu gestalten. Eine davon soll hier vorgestellt werden.

Geben Sie das gleich folgende Listing ein, speichern Sie es und führen es dann beispielsweise Ihrem Freund vor. Tippen Sie RUN ein und drücken Sie anschließend RETURN. Das Programm läuft wie gewöhnlich.

Der Spaß beginnt jedoch, wenn Sie Ihren Freund auffordern, es laufen zu lassen. Das Programm gibt sich plötzlich "bockig" und blockiert die Tastatur. Alle Versuche, es wieder neu zu starten, scheitern. Nur Sie haben die Gewalt über den Computer und schaffen es, die Tastatur wieder zugänglich zu machen.

Hier kommt nun dieses "mysteriöse" ESP-Programm:

```
5 rem esp-programm kap.3 vers.2
10 print"■"
20 r=53281
30 print "■"
100 for a=1 to 200
110 next a
115 print"■"
120 if peek(197)<> 1 then 300
150 :
200 poke r,0
205 for a=1 to 1e3: next a
207 print"■":x=16:y=10:gosub 500
210 print "Hallo!!"spc(67)"Ich gehorche nu
r dir!"
230 rem poke 197,64
270 if peek(197)=64 then poke r,5:poke r,9
:poke r,12:poke r,0:goto 270
280 goto 200
290 :
300 x=16:y=10:gosub 500
310 print "baeh!!"
315 print:print spc(9)"Ich kenne dich nich
t!!": print: print: print
320 print "ready."
330 poke 204,0: rem cursorzeichen an
340 if peek(197)<>55 then 340
350 fora=1 to 2:print "■";:next a
360 poke 198,0:rem tastaturbuffer leeren
365 poke 207,0:poke 204,1:rem cursor aus
370 print "■": goto 200
380 :
500 poke 211,x
510 poke 214,y
520 sys 58640
530 return
```

Wie funktioniert es? Natürlich hat es hier nichts mit übersinnlichen Kräften zu tun, sondern es gibt eine ganz logische Erklärung für das Verhalten des Programms, die auf einer simplen und dennoch effektvollen Programmierung beruht.

Nachdem Sie RUN eingegeben haben, muß die RETURN-Taste ein klein wenig länger als üblich gedrückt werden. Die Zeile 120 überprüft dies im Programm.

Bedienen Sie persönlich das Programm, so gelangen Sie in den Block von Zeile 200 bis 290. Dieser bewirkt eine effektvolle Animationsdemonstration. Ihrem Freund, der das Programm versucht, zum Laufen zu bringen, dagegen wird ein Systemcrash vorgetäuscht.

Nach der Meldung

```
Was willst du ?
Ich kenne dich nicht!

ready.
_
```

blinkt der Cursor normal weiter, doch sind alle Tasten blockiert. Nur Sie können das Programm mit SHIFT und M wieder reaktivieren.

Das Programm ist LIST-geschützt. Wenn Sie Ihre Freunde lange genug verwirrt haben, können Sie die ^-Taste betätigen und das Programm auflisten. Mit etwas Übung und schauspielerischem Geschick können Sie mit dem ESP-Trick selbst Programmier-Experten bluffen.

Dies ist jedoch nur eine mögliche Version eines ESP-Programms. Mit etwas Kreativität können Sie nach diesem Beispielmuster auch noch komplexere Versionen erstellen.

1.3 Vor-Definition von Variablen

Normalerweise ist es nicht erforderlich, Variablen zu definieren, bevor sie verwendet werden. Dies ist auch eine Besonderheit von BASIC.

Fast alle Programmiersprachen, wie beispielsweise PASCAL oder 'C', benötigen eine Variablendefinition.

Was bisher relativ unbekannt war: Auch BASIC ermöglicht derartige Definitionen. Sie werden durch den DIM-Befehl realisierbar.

Eine Beispieldefinition kann folgendermaßen aussehen:

```
10 DIM A,B,E1,E2
20 DIM Z%,WE%,A$,J$
```

Was bringt nun eine derartige Definition? Auf den ersten Blick erscheint sie ja überflüssig. Es ist jedoch manchmal effektiver, Variablen in einer bestimmten Reihenfolge im Speicher anzuordnen, besonders wenn die Ausführungsgeschwindigkeit kritisch ist. Dies gilt speziell für komplexere Programme. Hier sollten vor allem besonders häufig verwendete Variablen definiert werden.

Ein weiterer großer Vorteil von dimensionierten Variablen liegt jedoch auch in der Möglichkeit des strukturierten Programmierens. Sicher ist es Ihnen auch schon vorgekommen, daß Sie in einem Programm den Überblick über die verwendeten Variablen und deren Bedeutung völlig verloren haben.

Ein deutliches Anzeichen dafür sind auftauchende Fragen wie

```
...Wie lautete noch die High-Score Variable?...
...und was bewirkte denn aw$ ?... Habe ich die Variable z$
eigentlich schon für einen anderen Zweck belegt? ...
```

Eine derartige Konfusion läßt sich mittels der Variablendefinition beseitigen. Sie schaffen sich Übersicht, wenn alle in einem Programm verwendeten Variablen im Programmkopf deklariert werden.

1.4 Ausschalten des Bildschirms

Nein, sagen Sie bitte jetzt nicht: "Das mache ich doch immer über den Aus-Schalter am TV/Monitor". Das mag zwar stimmen, aber darum soll es uns hier nicht gehen. Sicher haben Sie festgestellt, daß der Bildschirm beim Laden von Kassettenprogrammen völlig ausgeschaltet, aber intern nicht gelöscht wird.

Uns soll es nun darum gehen, wie dieses programmgesteuert ablaufen kann. Das Abschalten des Bildschirms kann oft große Vorteile mit sich bringen, auch wenn Sie Ihre Programme nicht über die Datasette laden. Mögliche Anwendungsgebiete sind das Abschalten des Displays beim Laden von Diskette. Auch können Sie den Schirm einfach löschen, wenn beispielsweise ein fremder Benutzer Ihres Programms ein erforderliches Password nicht korrekt eingegeben hat.

Sehr sinnvoll ist diese Operation auch, wenn Sie den Sound verbesssern wollen. Vor allem, wenn es auf sehr feine Tonunterschiede, wie beispielsweise bei Speech-Synthesizern, ankommt, lohnt es sich oft, den Bildschirm auszuschalten und dadurch den Sound zu optimieren.

Sollten Sie ein 1540 Disk Drive besitzen und wollen sie dies als Zweit-Disk zur 1541 verwenden oder ist die 1540 das einzige Laufwerk, so ist es übrigens unbedingt notwendig, während der Disk-Operationen den Bildschirm auszuschalten. Nur so sind 1540 und C-64 bzw 1540 und C-128 kompatibel.

Kommen wir nun zu der Realisierung des Bildschirmschalters. Das Ausschalten geschieht über den POKE-Befehl...

```
POKE 53265,11
```

... und das Reaktivieren des Bildschirms über...

```
POKE 53265,27
```

1.5 C-Print-Befehl

Befassen wir uns nun mit der Realisierung einer Circle-Print-Routine. Wie der Name schon ausdrückt, können Sie mittels dieser beliebige Texte in einem frei wählbaren Radius kreisförmig auf dem Bildschirm ausgeben.

Geben Sie das Programm auf der folgenden Seite ein und bestimmen Sie dann als erstes den kreisförmig anzuzeigenden Text. Dann müssen Sie den Radius festlegen. Wählen Sie hier als Anfangswert zehn. Anschließend fragt Sie die Routine nach der Anzahl der darzustellenden Kreise. Geben Sie hier mehr als einen einzigen an, so wird der Radius nach jedem Durchgang automatisch reduziert. Der zweite Kreis ist also kleiner als der erste.

Hier nun unsere C-Print-Routine:

```
5 rem c-print
10 input "text";t$
15 input "radius 1-11";r
17 input "anzahl";m
20 print"■"
30 : for a=1 to m
40 :    for b=0 to 2 step 1.8/r
50 :     for c=1 to len(t$)
60 :       h=(b+c/20)*■
70 :       x=16+r*sin(h)
80 :       y=11-r*cos(h)
90 :        poke 211,x
100 :       poke 214,y
110 :       sys 58640
120 :      print mid$(t$,c,1)
130 :    next c
150 :  next b
160 :  r=r-sgn(sin(a+1))
170 :next a
```

Experimentieren Sie ruhig ein wenig mit der Routine; sie läßt sich auch zur Animation einsetzen.

Setzen Sie beispielsweise

```
140 PRINT "CLR"
```

in das Programm ein, so wird die anzuzeigende Meldung auf dem Display am Kreisradius entlang routiert.

1.6 Strukturen

Zum übersichtlichen und strukturierten Programmieren gehört einfach die Programmstrukturierung, d.h. das Einrücken von BASIC-Zeilen.

Das Commodore-BASIC tut sich hierbei jedoch etwas schwer. Im Gegensatz zu fast allen anderen BASIC-Versionen überliest der Commodore das führende Leerzeichen zwischen der Zeilennummer und dem ersten BASIC-Befehl. Das führt zu einem typischen Commodore-Programmierstil, bei dem Programme fast immer in einem Block erstellt werden.

Es ist jedoch von erheblichem Vorteil, wenn Programme strukturiert werden. Sie können so auf einen Blick erfassen, welche Schleifen zusammenhängen, wie Programmblöcke zueinandergehören und welche Sprünge zu welchen Zeilen gehören. Sie erzielen dadurch einen erheblichen Gewinn an Übersichtlichkeit.

In diesem Fall werden die Programme durch die Integration von Doppelpunkten direkt nach der Zeilennummer strukturiert.

Ein Beispiel dafür lautet:

```
10 FOR A=1 TO 20
20 :    FOR B=3 TO 6
30 :       C=B*A
40 :       PRINT C^2
50 :    NEXT B
60 NEXT A
```

Nach den Doppelpunkten können nun beliebig viele Leerzeichen zur Strukturierung folgen.

Es gibt jedoch noch eine elegantere Lösungsmöglichkeit. Dabei können Sie statt eines Doppelpunktes ein Graphikzeichen an den Zeilenanfang stellen. Der Interpreter überliest diese Zeichen einfach. Sie werden also im Listing nicht aufgeführt, wohl aber gespeichert. Nach dem Graphikzeichen können Sie anhand von dann folgenden Leerzeichen das Programm strukturieren.

1.7 Space-Tracer

Die oben aufgeführte Methode hat den bei umfangreichen Programmen unerwünschten Nebeneffekt, daß die eingefügten SPACE-Zeichen zusätzlichen Speicherplatz belegen. Wenn Sie ein speicherplatzkritisches Programm entwickeln, ist es also von Vorteil, alle überflüssigen Leerzeichen zu löschen. Dieser Vorgang sollte jedoch erst nach der Fertigstellung eines Programms erfolgen.

Wenn Sie ein Programm nun auf derartige speicherplatzverbrauchende Leerzeichen untersuchen wollen, so verwenden Sie dazu folgenden Poke-Befehl

```
POKE 129,58
```

Dieser bewirkt, daß alle Zeilen, die ein nicht in Anführungszeichen stehendes Space-Zeichen enthalten, einen Error generieren. Dadurch erhalten Sie die Möglichkeit, ein Programm auf Leerzeichen abzusuchen. Wenn Sie glauben, bereits manuell alle Spaces gelöscht zu haben, können Sie so bisher übersehene Zeichen erfassen.

Anhand der mit der Fehlermeldung zusammen ausgegebenen Zeilennummer können Sie so über den Poke-Befehl die betreffende Zeile nachträglich ändern, d.h. das entdeckte, überflüssige Leerzeichen löschen.

Mittels des Befehls

```
POKE 129,32
```

wird der Normalzustand wieder hergestellt.

1.8 Makros mit dem V2.0-BASIC

Sie wissen bereits, daß in BASIC mittels der DEF-FN-Konstruktion effektiv Funktionen vereinbart werden können.

Ein Beispiel dafür ist:

```
10 DEF FN FU(X)=SIN(X)*COS(Z)+2
20 Z=2
30 PRINT FN FU(.4)
```

Hier wird die mathematische Funktion

```
y = sin(x) * cos(z) + 2
```

im Programmkopf definiert und kann dann im folgenden Programm immer wieder und mit unterschiedlichen Parametern mittels FN FU(X) aufgerufen werden.

Die DEFine-Struktur ist jedoch noch erheblich leistungsfähiger und läßt sich zur Realisierung von Makros verwenden. Makros finden in vielen Programmiersprachen, wie beispielsweise der modernen Sprache 'C' universelle Verwendung. Sie sind im Gegensatz zu dem obrigen Funktionsbeispiel erheblich vielseitiger und universeller, da sie nicht nur auf mathematische Funktionen begrenzt sind. Der große Vorteil von Makros ist, daß ganze Ausdrücke als Substituenten in einem Programm behandelt werden.

In BASIC sind Makros an sich nicht möglich. Daß sie sich doch zu einem großen Grade realisieren lassen, zeigen wir auf den folgenden Seiten an einer kleinen BASIC-Makro-Bibliothek.

Sie werden an den Beispielen sehr schnell sehen, wie Makros in BASIC Problemstellungen erheblich vereinfachen und auf sehr elegante Weise bewältigen können. Makros dienen damit zum effektiveren Programmieren und sollten vielfältige Anwendung in Ihren Programmen finden.

Die hier vorgestellten Makros sind speziell für praktische Anwendungen konzipiert. Das bedeutet, daß Sie diese direkt übernehmen und in Ihren Programmen ohne Änderungen gewinnbringend einsetzen können.

1.8.1 Zentrierter Ausdruck

Ein zentrierter Ausdruck bewirkt, daß Texte genau in der Mitte einer Zeile augegeben werden. Ein derartiges Display sieht beispielsweise folgendermaßen aus:

Computern und Hacken
ist
Fun, Fun, Fun!

Die allgemein übliche Methode, einen zentrierten Ausdruck zu erzeugen, ist meistens, Leerzeichen in den PRINT-Befehl zu integrieren.

Das artet jedoch immer in einem wüsten Probieren aus, um festzustellen, wann der zu PRINTende Text sich genau in der Mitte des Bildschirms befindet. Zum anderen verschwenden die zusätzlichen SPACEs unnötigen Speicherplatz. Mittels eines Makros läßt sich ein zentrierter Ausdruck ohne jeglichen Aufwand völlig automatisch ausführen.

Unser Makro lautet dazu:

```
10 DEF FN ZN(Y) = 20 - (LEN(T$)/2)
```

Um nun beispielsweise unseren dreizeiligen Text von der vorigen Seite zentriert auszugeben, müssen die folgenden Zeilen eingegeben werden:

```
100 T$="Computern und Hacken": PRINT TAB(FN ZN(Y));T$
110 T$="ist": PRINT TAB(FN ZN(Y));T$
120 T$="Fun, Fun, Fun!!": PRINT TAB(FN ZN(Y));T$
```

Der in der Variable T$ enthaltende Text wird über FN ZN(Y) automatisch zentriert. Das Argument Y wird dabei nicht verwendet, ist jedoch zur Vermeidung eines Syntax-Errors erforderlich.

1.8.2 Logarithmus

Die Funktionen LOG(X) sowie EXP(X) des BASICs arbeiten im natürlichen Logarithmus. Für viele mathematische Anwendungen wird jedoch ein 10er Logarithmus benötigt. Dieser läßt sich

mittels eines Makros sehr einfach realisieren. Unser Ausdruck lautet dabei:

```
10 DEF FN LG(X)= LOG(X)/LOG(10)
```

Der 10er Logarithmus der Zahl 7.21 wird dann so bestimmt:

```
20 PRINT FN LG(7.21)
```

Das LG Ergebnis lautet

0.857935, der natürliche Logarithmus dagegen
1.975469.

Die Schwesterfunktion von LOG(X), EXP(X), kann mittels der eben beschriebenen Methode ebenfalls als 10er Funktion definiert werden. Der entsprechende Makro dazu lautet:

```
10 DEF FN EX(X)= EXP(LOG(10)*X)
```

Um nun beispielsweise EX(5) zu bestimmen, geben Sie einfach den normalen Funktionsaufruf

```
PRINT FN EX(5)
```

ein.

Das Ergebnis ist 100000, was natürlich 10 hoch 5 entspricht.

1.8.3 Modulo

FORTRAN, PASCAL und 'C' besitzen eine sog. Modulo-Funktion. Diese wird auch oft in BASIC benötigt. Hier hilft ebenfalls ein Makro, um sie zu implementieren:

```
10 DEF FN MO(X)= INT((A/B-INT(A/B))*B+.5)
```

Die MOD-Funktion liefert den Rest der Division von A durch B.

```
20 MOD 3
```

liefert also 2, nämlich den Rest der Division von 20/3.

Mittels unseres Makros sieht die Anwendung fogendermaßen aus:

```
20 A=20: B=3
30 PRINT FN MO(X)
```

Indem Sie FN MO(X) einsetzen, haben Sie an jeder Stelle Ihres Programms die Zugriffsmöglichkeit auf die Modulofunktion.

1.8.4 Deg-Rad-Umwandlung

Das BASIC des Commodores berechnet trigonometrische Funktionen in Radians und nicht in Degrees (Grad). Diese immer wieder für mathematische Programme benötigte Umwandlung läßt sich mittels eines verschachtelten Makros realisieren.

Der Makro sieht folgendermaßen aus:

```
10 DEF FN DG(X)=X*(PI, durch Shift ^) / 180
20 DEF FN SI(X)=SIN(FN DG(X))
30 DEF FN CO(X)=COS(FN DG(X))
40 DEF FN TA(X)=TAN(FN DG(X))
```

Die DEGree-Umwandlungsfunktion FN DG(X) wird dabei in den neuen Sinus, Cosinus und Tangensfunktionen aufgerufen.

Wollen Sie nun den Wert von SIN(x) in DEG(Grad) statt in RAD erhalten, so geben Sie beispielsweise ein:

```
PRINT FN SI(.5)
```

Das Ergebnis lautet 0.0087265.

Sie sollten anhand unserer bisherigen Beispiele einen Eindruck gewonnen haben, wozu sich Makro-Simulationen einsetzen lassen und wie Sie Ihre eigenen Makros kreieren können.

1.8.5 Maxi-Division auf alle Stellen

Bis zu einem gewissen Grad lassen sich mittels der Define-Konstruktion auch komplexe Makros gestalten. Ein Beispiel stellte die Realisierung der RAD-DEG-Umwandlung mittels eines verschachtelten Makros dar. Ein Beispiel für derartige komplexe Makros stellt eine Divisionsroutine dar, welche die Division mit bliebiger Genauigkeit, d.h. beliebiger Nachkommastellenzahl ausgibt.

Eine solches Makro kann in BASIC nur über eine Subroutine realisiert werden:

```
100 PRINT INT(A/B)".";
110 H=A-B*INT(A/B)
120 FOR I=1 TO S
130 :  PRINT INT(H*10/B)"(Cursor Left)";
140 :  H=10*H-B*INT(H*10/B)
150 NEXT I
160 RETURN
```

Diese Routine wird folgendermaßen aufgerufen:

```
A=1: B=7: S=20: GOSUB 100
```

Die Variable A stellt dabei den Zähler, B den Nenner der durchzuführenden Division dar. Mit S bestimmen Sie die Anzahl der Nachkommastellen und mittels GOSUB 100 wird die Routine aktiviert.

1.8.6 Fakultät

Die Fakultätsfunktion ist in BASIC nicht vorhanden, wird jedoch des öfteren benötigt. Sie ist ein weiteres Beispiel eines Makros, der nur über eine Subroutine bewältigt werden kann:

```
100 FOR I=1 TO Z: F=F+LOG(I):NEXT I
110 F=F/LOG(10)
120 PRINT 10^(F-INT(F))"E"INT(F)
```

Wollen Sie beispielsweise 60! bestimmen, so müssen Sie einfach

```
Z=60: GOSUB 100
```

eingeben und erhalten als Ergebnis 8.32098688 E81.

Diese beiden letzten Routinen sind wie bereits angesprochen, eigentlich keine Makros, sondern Makro-Subroutinen. Komplexe Makros lassen sich in BASIC leider nur über Unterprogramme realisieren bzw simulieren. Echte Makros sind in BASIC, wie in unseren ersten vier Beispielen gezeigt, mittels der DEFine-Konstruktion äußerst elegant realisierbar und sollten auch in Ihre Programmierpraxis Eingang finden.

1.8.7 DEF FN X(Y) - Das unbekannte Wesen

Viele Programmierer ziehen es vor, an Stelle dieser eigentlich komfortablen Funktion - zugegeben, sie ist in den Handbüchern gewöhnlich unvollkommen (bisweilen auch unverständlich) beschrieben - einige Programmzeilen mehr, oft auch Subroutinen einzugeben.

Außer der erwähnten schlechten Dokumentation muß es dafür noch einen weiteren Grund geben: Die Funktion scheint nicht sehr mächtig zu sein, jedenfalls nicht auch den ersten Blick (der leider oft entscheidet über Brauchbarkeit und Unbrauchbarkeit), da offensichtlich nur ein Argument an die Funktion übergeben werden kann.

Wir wollen Ihnen deshalb in diesem Abschnitt zum einen den Umgang mit dieser Funktion veranschaulichen, zum anderen zeigen, daß auch recht komplexe Formelgebilde mit mehreren Variablen durch Schachtelung mehrerer DEF FNX(Y) gelöst werden können.

Eine Funktionsdefinition ist folgendermaßen aufgebaut:

```
DEF FN Name (Funktionsvariable) = arithmetischer Ausdruck
```

z.B.

```
DEF FN A(X) = 2*X + B
```

Unsere Funktion hat den Namen A und die Funktionsvariable X. Wird die Funktion nun aufgerufen, so wird statt des X der Wert verwendet, der beim Aufruf übergeben wird. Eventuell existierende Variable mit dem Namen X und A bleiben dabei unverändert. Die Funktionsvariable X dient nur als Platzhalter für den aktuellen Wert beim Funktionsaufruf. Im Gegensatz dazu bezieht sich B auf die Variable B, die vorher definiert wurde. Die Funktionsvariable wird auch oft als Dummy-Variable bezeichnet. Das Ergebnis einer Funktion muß immer numerisch sein, ein Stringausdruck ist dabei nicht erlaubt.

Zur Einstimmung ein einfaches, oft benötigtes Beispiel, nämlich die kaufmännische Rundung. Wie Ihnen sicher bekannt ist, wird hierbei die dritte Dezimalstelle eines DM-Betrages derart behandelt, daß bei Werten von kleiner 5 abgerundet und bei größer 4 aufgerundet wird.

Die meisten Programmierer führen diese Rundung direkt am Ort des Entstehens in der Form

```
A = INT (B*100+.5)/100
```

durch. Geschieht das öfter, kann man bei Ersatz dieser Formel durch

```
A = FNX(B)
```

viel Platz sparen. Konkret sieht das Ganze so aus:

```
10 DEF FN K(B) = INT(B*100+.5)/100
...
100 A = FNK(B)
```

Den Befehl in Zeile 100 können Sie nun so oft wie nötig für die längere Formel einsetzen, wobei die Variable B den ungerundeten Wert enthalten sollte.

Nun ein Beispiel für die Schachtelung von Funktionen. Hier soll anhand des Netto-Einkaufspreises der Verkaufspreis unter Zugrundelegung der gewünschten Handelsspanne und des Mehrwertsteuersatzes ermittelt werden.

C ist der Einkaufspreis
H die Handelsspanne in % "von oben"
M der Mehrwertsteuersatz in %

```
10 DEF FN V(C) = (C/(1-H/100)*(1+M/100)
20 DEF FN K(B) = INT (FN V(C)*100+.5)/100
...
100 A = FN K(C)
```

A enthält nach Zeile 100 den Verkaufspreis, gerundet und inklusive Mehrwertsteuer. Natürlich können Funktionen noch tiefer geschachtelt werden. Dabei ist eine maximal 10fache Schachtelung möglich. Versucht man noch tiefer zu schachteln, so erhält man einen 'OUT OF MEMORY ERROR', da die Zwischenergebnisse im Stack gespeichert werden, der nur eine begrenzte Größe hat. Sind während des FN-Aufruf noch Unterprogramme (GOSUB) oder Schleifen (FOR-NEXT) aktiv, die ebenfalls den Stack benötigen, so ist nur eine entsprechend geringere Schachtelung möglich, die jedoch für praktische Anwendungen immer noch genügend sein dürfte.

1.9 Die Anti-Error Routine

Es ist immer wieder frustrierend, wenn beim Lauf eines mit viel Mühe selber entwickelten Programms dieses plötzlich "abstürzt".

Die hier vorgestellte Routine registriert jeder Error-Typ und bricht ein Programm dann nicht wie gewöhnlich ab, sondern springt automatisch eine sogenannte Error-Seite eines Programms an.

Fast jedes Programm enthält versteckte Fehlerquellen, von denen sich viele nur mit einem erheblichen Aufwand, oft aber überhaupt nicht eleminieren lassen.

Da ist beispielsweise das Disk-Drive nicht angeschaltet, der Nenner eines Bruches wird aus unerwarteten Gründen plötzlich null, auf einmal entsteht ein OUT OF MEMORY Error, ein Re-DIM wird ausgeführt, Files sind bereits mit OPEN eröffnet oder aber geschlossen, VERYFY oder LOAD erzeugt einen Fehler und und und.

Feststeht, die möglichen unerwarteten Fehlerquellen sind vielfältig.

Hier schafft die ANTI-ERROR-Routine radikal reinen Tisch und Abhilfe. Statt eines Programmabsturzes, welcher einen fremden Anwender Ihres Programms mit einer Error-Meldung und einem munter blinkenden Cursor hoffnungslos im Stich läßt, bewirkt die neue, vielseitige Routine eine absolute Fehlersicherung. Das Programm stürzt nicht mehr ab, sondern der Anwender erhält eine detailierte Fehlerbeschreibung, gleichzeitig läuft das Programm jedoch wie bisher weiter. Der unerschätzbare Nutzen dieser Routine liegt auf der Hand. Befassen wir uns nun mit der Realisierung. Geben Sie zunächst das folgende Programm ein:

```
60000 POKE 53281,0
60010 T$="G"+CHR$(13)
60020 POKE 679,134
60030 POKE 680,10:REM ERROR-MELDUNG
60040 POKE 681,224:POKE 682,128
60050 POKE 683,240:POKE 684,34
60080 W=119
60100 FOR A=685 TO 685+LEN(T$)*5-5 STEP 5
60110 :  POKE A,169
60120 :  POKE A+1,ASC(MID$(T$,W-118,1))
60130 :  POKE A+2,141:POKE A+3,W
60135 :  W=W+1
60140 :  POKE A+4,2
60150 NEXT A
60200 POKE A,169:POKE A+1,6
60210 POKE A+2,133:POKE A+3,198
60230 POKE A+4,76:POKE A+5,139
60240 POKE A+6,227
60300 POKE 768,167:POKE 769,2
```

Die ANTI-ERROR-Routine bewirkt, daß der Inhalt der Variablen T$ in Zeile 60010 in den Tastaturbuffer übertragen wird. In unserer Beispielroutine ist dieser Inhalt

g(Shift O)200 +Return

Das bedeutet, daß bei einem beliebigen Error die Zeile 200 über GOTO angesprungen wird.

Das Programm ist jedoch sehr flexibel angelegt. Sie können auch andere Befehle wie beispielsweise

```
list

print2*5
```

oder

```
run
```

in T$ ablegen können. Am sinnvollsten ist jedoch ein GOTO-Befehl, gefolgt von der Zeilennummer, in der Sie eine Error-Seite gestalten können. GOTO hat hier den Vorteil, daß bekannterweise die Variableninhalte erhalten bleiben.

Testen Sie nun ruhig die einzelnen Fehlertypen durch. Egal, welchen Error Sie auch erzeugen, die ANTI-ERROR Routine springt immer sicher die Error-Seite an.

Geben Sie beispielsweise manuell die folgenden Operationen ein:

```
PRINT 1/0          ... division by zero
PRINT 6^99         ... overflow error
QWERT              ... syntax error
DIM A(8):DIM A(8)  ... redim'd array
DIM A(40000)       ... illigal quantity error
```

Genauso wie diese manuellen Errors werden natürlich auch in einem Programm erzeugte Fehler von der Routine abgefangen.

Sie können die ANTI-ERROR-Routine natürlich auch abschalten und den normalen Error-Modus wiederherstellen.

Setzen Sie die in der Zeile 60300 festgelegten Vektoren wieder auf die alten Werte. Geben Sie dazu ein:

```
POKE 768,139
```

und

```
POKE 769,227
```

Ich bin sicher, daß Sie diese Routine in fast jedem Ihrer Programme effektiv einsetzen können. Die dadurch gewonnene Absturzsicherheit und Benutzerfreundlichkeit erhöhen die Qualität jedes Programms.

1.10 Die 3 Uhren des C-64

Es ist erstaunlich, wie wenig auch den versiertesten Programmierern über die Zeitfunktionen des C-64 bekannt ist. Der Commodore besitzt in nämlich drei(!) Time-Routinen, die alle voneinander völlig unabhängig sind.

Wie diese zu adressieren sind, wie sich deren Genauigkeit beträchtlich erhöhen läßt, was es mit Delay-Funktionen auf sich hat, wie sich die normale TI$-Clock auf 1/100 Sekunden "tunen" läßt erfahren Sie im Folgenden.

Befassen wir uns im folgenden zuerst mit der sog. JIFFI-CLOCK. Diese mißt die Zeit bekanntlich in 60stel Sekunde. Natürlich wissen Sie bereits lange, daß Sie eine einfache Uhr mittels der Variablen TIME$ realisieren können.

Ein einfaches Beispielprogramm sieht folgendermaßen aus:

```
10 TI$="000000"
20 FOR I=0 TO 1 STEP 0: PRINT "(Cursor hoch)";TI:NEXT I
```

Diese Uhr zählt auf 1/60 Sekunden genau. Nur ist die Basis 60 der JIFFI-CLOCK nicht so ganz glücklich gewählt:

Wenn Sie das Programm laufen lassen und nach einiger Zeit mittels STOP-Taste beenden, erhalten Sie beispielsweise als letzte Zahl

```
13082
```

Diese stellt die Anzahl der 1/60 Sekunden dar, die die Uhr lief. Um hier auf die Zahl der Sekunden zu kommen, müssen Sie die 13082 durch 60 teilen. In unserem Fall erhalten wir 218 Sekunden.

Auf ähnliche Weise wird die Anzahl der Minuten bestimmt. Sie müssen dabei 13082/(60*60) teilen, was ein Ergebnis von 3.6 Minuten erzeugt.

Diese Umwandlungen lassen sich natürlich auch über ein Makro vornehmen:

```
10 DEF FN MI(X)=TI/360
```

Trotzdem ist jede derartige Berechnung aufwendig. Es geht auch ganz einfach.

Geben Sie die folgenden beiden POKEs ein:

```
POKE 56324,242
```

und

```
POKE 56325,39
```

Sie bewirken, daß die JIFFI TI-Zeitvariable nicht mehr wie bisher jede 60stel Sekunde, sondern jetzt genau jede 100stel Sekunde um eins erhöht wird.

Dieses wird sofort am geänderten Verhalten Ihres Computers ersichtlich. Alle Vorgänge, die auf die Zeitfunktion zugreifen, wie beispielsweise die Blinkfrequenz des Cursors, erfolgen nun mit einer um den Faktor 1.6666 erhöhten Geschwindigkeit.

Der große Vorteil dieser so geänderten Uhr liegt darin, daß Sie so eine Stoppuhr mit einer Anzeigegenauigkeit von 1/100stel Sekunden realisiert haben. Die Beispieluhr zeigt Ihnen nun also die hundertstel Sekunden an, die verstrichen sind, nachdem TI$ mit "000000" initialisiert wurde.

Um die Anzahl der ganzen Sekunden zu erhalten, muß natürlich noch der Wert der Variablen TI durch 100 geteilt werden.

1.11 Adressierung der internen "TOD"-Clock

Die zweite Uhr ist die sogenannte TOD-CLOCK. Diese gibt die Time Of Day, die Tageszeit an. Kommen wir erst einmal zu den Vor- und Nachteilen der TOD- und der JIFFI-CLOCK. Ein großer Nachteil der JIFFI-CLOCK ist ihre Ungenauigkeit. Diese ist besonders dadurch bedingt, daß die JIFFI-CLOCK bei I/O-Operationen einfach ausgeschaltet wird. Das wirkt sich besonders nachteilig aus, wenn Sie beispielsweise viele Disk-Zugriffe ausführen. Die Zeit wird dabei äußerst ungenau.

Die TOD-CLOCK ist dagegen völlig autonom. Unterbrechungen sind bei ihr nicht möglich. Des weiteren ist die TOD-Zeit völlig unabhängig von JIFFI-CLOCK, d.h. Sie haben damit die Möglichkeit, beide Uhren mit völlig unterschiedlichen Zeiten zu belegen.

Es gibt bei allen Vorteilen einen Nachteil. Und das ist die etwas aufwendige Programmierung von TOD. Die Initialisierung und der Zugriff auf die TOD-Zeit ist lange nicht so einfach wie die der JIFFI-CLOCK über die TI-Variable.

Geben Sie nun zunächst das Listing der TOD-CLOCK ein:

```
10 poke 53281,0: poke 53272,23
20 print"■■":print" T O D - CLOCK":print:p
rint:print
30 print" TOD-Zeit und zum  Vergleich JIFF
I-Zeit  ":print:print:print:print
100 cl=56331
110 ic=56333
120 bc=56335
130 poke bc,peek(bc) and 127
200 :
210 for i=0 to 3
220 :  poke cl-i,0
230 next i
240 :
400 :
500 h=peek(cl)
510 m=peek(cl-1)
520 s=peek(cl-2)
530 t=peek(cl-3)
600 :
900 print"■";
1000 print" z= "chr$((16 and h)/16+48);
              chr$((15 and h)+48)":";
1010 printchr$((240 and m)/16+48);
        chr$((15 and m)+48)":";
1020 printchr$((240 and s)/16+48);
        chr$((15 and s)+48)".";
1030 printchr$(t+48)" "chr$(65-(h>127)*15)
"m";
1100 print"    JIFFI-Zeit: ";ti$
1500 goto 500
```

Der Programmablauf sieht folgendermaßen aus:

- T O D - CLOCK
- TOD-Zeit und zum Vergleich JIFFI-Zeit
- z= 00:00:12:9 am JIFFI-Zeit: 001449

Sie sehen hier, daß die beiden vergleichsweise ausgegebenen Zeiten wirklich unabhängig voneinander sind.

Funktion der TOD-Clock

Zunächst werden in den Zeilen 100 bis 120 die CLOCK-Register gesetzt. Dabei beinhaltet die Adresse

56331 die Stunden,
56333 die Minuten

und

56335 die Sekunden

der TOD-Clock.

Sicherheitshalber wird dann das Control Bit der Uhr in der Zeile 130 auf Null gesetzt, um sie so schreib-bereit, d.h. Write-fähig zu machen.

Die FOR-NEXT-Schleife in Zeile 200 bis 230 initialisiert die Zeit. Sie entspricht also der Anweisung

```
TI$="000000"
```

bei der JIFFI-CLOCK.

Nachdem die Initialisierung abgeschlossen ist, werden die Stunden-, Minuten- und die Sekundenregister in den Zeilen 500 bis 530 gelesen. Diese aktuellen Werte werden dann in der Zeile 1000 bis 1030 auf dem Display ausgegeben. Dieser Lese- und Ausgabevorgang wird dann permanent wiederholt.

Um Skeptikern zu beweisen, daß die JIFFI- und TOD-Zeit wirklich völlig unabhängig voneinander sind, werden zum Vergleich noch parallel die beiden Zeiten nebeneinander angezeigt.

Da JIFFI nicht initialisiert wurde, gibt sie die Zeitspanne an, die seit dem Einschalten des Computers vergangen ist.

Zugriff auf die dritte interne Clock

In den vorigen Abschnitten haben wir uns ausführlich mit zwei der drei internen Uhren des C-64 befaßt. Kommen wir nun zu der Initialisierung und Adressierung der dritten Clock.

Wenn Sie bereits das Listing der TOD-CLOCK eingegeben und erste Erfahrungen gesammelt haben, so wird Ihnen das Verständnis der neuen Zeitmessung keine Schwierigkeiten bereiten.

Diese dritte Uhr ist nämlich ebenfalls eine TOD-CLOCK. Die TOD-Uhr ist in den 6526-Chip integriert. Da Ihre Maschine zwei dieser Chips besitzt, verfügen Sie auch über zwei unabhängige TOD-CLOCKS. Wie Sie erste TOD-CLOCK adressieren, die sich in dem ersten 6526 befindet, die auch den Interrupt-Timer beinhaltet, haben wir bereits erörtert.

Nun, die Steuerung der TOD-CLOCK auf dem zweiten 6526 Chip fast genauso zu realisieren.

Dazu müssen Sie lediglich

den Wert 256

zu jeder relevanten POKE-Adresse der ersten TOD-Uhr addieren.

Achten Sie jedoch unbedingt darauf, das Controll-Register IC=56335 nicht zu ändern.

Unser neues Listing zur zweiten TOD-CLOCK sieht dann folgendermaßen aus:

```
10 poke 53281,0: poke 53272,23
20 print"■■":print" Zweite  T O D - CLOCK"
:print:print:print
30 print" TOD-Zeit und zum  Vergleich JIFF
I-Zeit  ":print:print:print:print
100 cl=56586
110 ic=56333
120 bc=56591
130 poke bc,peek(bc) and 127
200 :
210 for i=0 to 3
220 :  poke cl-i,0
230 next i
240 :
400 :
500 h=peek(cl)
510 m=peek(cl-1)
520 s=peek(cl-2)
530 t=peek(cl-3)
600 :
900 print"■";
1000 print" z= "chr$((16 and h)/16+48);
               chr$((15 and h)+48)":";
1010 printchr$((240 and m)/16+48);
        chr$((15 and m)+48)":";
1020 printchr$((240 and s)/16+48);
        chr$((15 and s)+48)".";
1030 printchr$(t+48)" "chr$(65-(h>127)*15)
"m";
1100 print"    JIFFI-Zeit: ";ti$
1500 goto 500
```

Eine kurze Anmerkung zu der Programmzeile 1030:. Diese stellt eine Besonderheit der beiden JIFFI-CLOCKs dar. Über die Adresse T+48 können die zehntel Sekunden direkt ausgegeben werden. Anschließend wird durch die letzte CHR$-Anweisung angezeigt, um welche Time Of Day es sich bei der aktuellen TOD-Zeit handelt.

Abhängig vom leitenden Bit der Adresse H wird dann P- oder A-M (PM/AM) ausgegeben, was durch die Anweisung

```
PRINT CHR$(65-(H>127)*15)"M";
```

bewerkstelligt wird.

1.12 Statement-Flasher

Viele BASIC-Versionen kennen einen FLASH-Befehl, welcher bestimmte Ausgaben blinkend auf dem Display anzeigt. Beim C-64, der dies bekanntlich nicht beherrscht, kann man sich jedoch schnell helfen. Geben Sie zur Veranschaulichung das folgende Programm ein:

```
10 PRINT SPC(5)"DRUECKEN SIE...": PRINT
20 PRINT " RETURN (reverse off) ZUM HAUPTMENU"
30 PRINT "(2 mal cursor hoch)"
40 IF PEEK(162) AND 16 THEN PRINT "(reverse on)";
50 :
60 GET X$:IF X$="" THEN 20
70 weiter...
```

Das Wort

```
" RETURN "
```

wird hier in einem FLASH angezeigt. Sie können dabei die FLASH-Wechselperiode beliebig verändern, indem Sie den Wert "16" der Zeile 40 durch 1,2,4,8,32,... ersetzen.

Die Adresse 162 enthält den Zähler der JIFFI-Clock (TI und TI$). Die Zeile 40 bestimmt über den aktuellen Wert die Blinkperiode.

1.13 Ein Reverse-Befehl

Im Modus CTRL 9 (REV ON) ausgegebene Großbuchstaben sind in der 8*8-Matrix des Commodore oben nicht umrandet. Dies kann man jedoch schnell beheben. Ein Beispiel dafür gibt das folgende Programm:

```
10 a$="(rev on) RETURN "
20 print a$: rem altes reverse
30 :
40 b$=chr$(164):b$=b$+b$+b$+b$
50 print b$;b$: rem verbessertes reverse
60 print a$
```

Wenn Sie Commodore-"@", also CHR$(164), eine Zeile über im REVERSE-Modus ausgegebenen Großbuchstaben eingeben, so ist das graphische Display professioneller und ansprechender.

1.14 VAR-Tracer

Solange Sie ein Programm nicht ganz genau planen, laufen Sie Gefahr, den Überblick über die verwendeten Variablen zu verlieren. Wenn Sie ohne zusätzlichen Aufwand die Variablenstruktur eines BASIC-Programms in den Griff bekommen wollen, so ist dies über einen VARIABLEN-TRACER realisierbar.

Dieser listet Ihnen alle verwendeten Variablen sowie die Variablentypen auf.

Die Routine besticht durch ihre Kompaktheit:

```
60000 FOR XV=PEEK(45) + 256*PEEK(46) TO PEEK(47)-
14 + 256*PEEK(48) STEP 7: IV$="_":REM SPACE
60010 BV$=CHR$(PEEK(XV+1)): IF ASC(BV$)>=128 THEN
BV$=CHR$(PEEK(XV+1)-128): IV$="$"
60020 AV$=CHR$(PEEK(XV)) : IF ASC(AV$) >=128 THEN
AV$=CHR$(PEEK(XV)-128 : IV$="%"
60030 PRINT CHR$(18);AV$;BV$;IV$ : NEXT XV: PRINT
CHR$(146);"OK!"
```

Fügen Sie die vier Zeilen an Ihr BASIC-Programm an und starten Sie die Routine mit

```
GOTO 60000
```

Beachten Sie dabei, daß alle vorher verwendeten Variablen mit den zugehörigen Typenindizis INTEGER "%", STRING "$" und FLOAT "_" angezeigt werden. Die in der VAR-TRACER-Routine verwendeten lokalen Variablen werden jedoch unterdrückt.

Die Utility untersucht also den Speicher auf die vereinbarten Variablen und gibt diese in derselben Reihenfolge aus, in der sie nacheinander mit Daten belegt wurden.

Der VAR-TRACER funktioniert folgendermaßen:

Alle Variablen, außer mittels der DIM-Konstruktion definierten Feldern, werden in dem VTL, dem List Table abgelegt.

Der VTL befindet sich im Speicher direkt nach dem BASIC-Programm. Die Anfangs- und End-Adressen des VTL werden in der Zeile 60000 bestimmt und legen die Variablen-Suchschleife fest.

Jeder Variableneintrag im VTL beträgt 7 Bytes. Die ersten zwei davon beinhalten den Variablennamen. Dieser wird in die beiden Variablen A$ und B$ eingelesen.

Die folgenden fünf Bytes beinhalten den Wert der Floatvariablen. Bei Integervariablen werden dagegen lediglich zwei weitere Bytes verwendet. Strings dagegen werden im Speicher in einem anderen Bereich abgelegt. Die jeweilige Adresse ist in dem vierten und fünften Byte abgelegt, das dritte Byte beinhaltet die Stringlänge.

Der Variablentyp, welcher durch die Variable IV$ repräsentiert wird, ist folgendermaßen bestimmbar: Ist der ASCII-Wert des ersten Bytes, also von der von A$, größer als 128, so handelt es sich um eine Integervariable.

Entsprechend gilt: Ist der zweite ASCII-Wert, der von B$, größer als 128, so ist die Variable vom Typ String.

Sind dagegen die Werte beider Variablen kleiner als 128, liegt eine Float-Variable vor.

Diese Untersuchungen werden in den Zeilen 60010 und 60020 vorgenommen und der Variablen IV$ der entsprechende Variablentyp ("$","%","_") zugewiesen.

1.15 Micro-Del-Routine

Ich stelle Ihnen hier die mit Abstand einfachste, kürzeste und verblüffendste LINE-DELETE-Routine vor.

Geben Sie dazu die folgenden beiden Zeilen ein:

```
60000 POKE 214,22:POKE 211,0:SYS 58640:PRINT I
60010 PRINT "I=";I+1;"J=";J;": IF I<J-1THEN 60000
(4 mal Cursor hoch)" :POKE 198,2:POKE 631,13:POKE632,13
```

Diese Mikro-Routine funktioniert folgendermaßen:

Angenommen Sie wollen alle Zeilen im Bereich zwischen 1000 und 1200 löschen, so müssen Sie die folgende Anweisung eingeben

```
I=1000: J=1200: GOTO 60000
```

Es ist schon erstaunlich, wie man mit einem Minimum an Aufwand einen großen Effekt erzielen kann.

Wie funktioniert die Routine? Angewendet wird hier der programmierte Direktmodus. Für alle, die diesen noch nicht kennen, das Wesentliche:

Alle Tasten, die Sie drücken, werden in dem Tastatur-Buffer gespeichert. Dieser belegt die zehn Speicherplätze von 631 bis 640. Der Buffer muß immer im Zusammenhang mit der Adresse 192 angewendet werden. Diese muß die Anzahl der bis zu maximal zehn in den Tastaturbuffer gePOKEten Zeichen enthalten.

Probieren Sie zur Veranschaulichung einmal dazu die folgende Zeile aus:

```
FOR A=631 TI 640: POKEA,113: NEXT
```

Dadurch wird der Tastaturbuffer mit zehn Graphikzeichen (CHR$(113)) belegt. Nun müssen Sie noch anschließend

```
POKE 192,10
```

eingeben, um den Inhalt des Buffers auf dem Display auszugeben.

In unserer Line-Delete-Utility wurde die Programmierung des Buffers folgendermaßen angewendet:

Zunächst wird die aktuelle Cursorposition in 60000 auf die Zeile 22 und die Spalte 0 festgelegt.

Wenn Sie nun beispielsweise als erste zu löschende Zeile 1000 und als letzte 1200 gewählt haben, erscheint die folgende Anzeige auf dem Bildschirm:

```
READY

_
1000
I= 1001 :J= 1200 :IF I<J-1 THEN 60000
```

Zunächst wurde dabei die erste zu löschende Zeilennummer, 1000, ausgegeben, anschließend die dann folgende Befehlszeile.

Die vier "Cursor hoch"-Zeichen nach der letzten Ausgabe (nach 60000) bewirken, daß die READY-Meldung in der Zeile 20 erscheint. Der Cursor steht dann darunter in Zeile 21.

Das Programm führt nun über die Buffer-Anweisungen in 60010 zwei CHR$(13), d.h. zwei RETURNs aus. Diese bewirken, daß die beiden vorher ausgegebenen Anweisungen in den Bildschirmzeilen 22 und 23 über den programmierten Direktmodus eingegeben und befolgt werden.

Dadurch werden alle Zahlen von 1000 bis 1200 so ausgegeben und die entsprechenden Zeilen gelöscht. Durch geschicktes Programmieren nach dieser Vorlage können Sie auch beliebige eigene Programme mittels des programmierten Direktmodus optimieren.

Die maximale Grenze liegt dabei bei zehn Programmanweisungen, die Sie zunächst untereinander auf dem Bildschirm ausgeben können und dann mittels zehn CHR$(13)-Anweisungen über den Buffer hintereinander ausführen lassen können.

1.16 Micro Auto-Line Utility

Mit nur wenigen Änderungen können Sie die obige Routine auch zu einer AUTO-LINE Routine umfunktionieren. Eine derartige Routine übernimmt für Sie das Eintippen der Zeilennummern. Sie müssen dann lediglich noch die BASIC-Befehle eingeben.

Die neue Routine sieht folgendermaßen aus:

```
61000 POKE 214,22:POKE 211,0:SYS 58640:PRINT I;":"
61010 PRINT "I=";I+S;"J=";J;"S=";S; " :IF I<J-1 THEN 60000 (4 mal
Cursor hoch)"
61020 POKE 198,2:POKE 631,13:POKE632,13
```

Die Anwendung läuft dabei folgendermaßen ab:

Wollen Sie beispielsweise ein Programm ab der Zeile 10 mit einem Abstand von jeweils fünf Zeilen (Schrittweite) mit voraussichtlich ca. 100 BASIC-Zeilen erstellen, so müssen Sie diese Anweisung eingeben:

```
I=10: J=1000: S=5: GOTO 61000
```

Dabei ist I die erste, J die letzte Zeile und S stellt die Schrittweite dar.

Das Resultat ist ein vollständig durchnumeriertes Leer-Programm von folgendem Format:

```
10 :
15 :
20 :
25 :

... etc.  ...

995 :
1000 :
```

Diese einzelnen Zeilen brauchen Sie nun lediglich noch mit den entsprechenden BASIC-Anweisungen zu belegen, das lästige Eingeben der Zeilennummern kann entfallen.

1.17 Aufspüren von 'Out of Stack Memory Errors'

Geben Sie einmal das folgende Programm ein und starten es mit RUN:

```
10 GOSUB 20
20 GOTO 10
```

Bereits nach Bruchteilen von Sekunden meldet sich der Computer mit einer für die meisten Programmierer völlig überraschenden

```
?OUT OF MEMORY ERROR IN 10
```

Meldung.

Warum entsteht nun diese scheinbar völlig unbegründete Fehlerausgabe? Mit der fehlerkorrigierten FRE()-Anweisung

```
PRINT FRE(0)-65536*(FRE(0)<0)
```

können Sie sich ja doch davon überzeugen, daß der Computer noch mehr als ausreichend Speicherplatz besitzt.

Die OUT OF MEMORY-Meldung bezieht sich in diesem Fall auf die Überfüllung des Stapels. Der Stapel stellt bekanntlich die Speicherzone von 256 bis 511 dar. In diesen Speicherbereich werden Hilfsanweisungen abgelegt.

Unser zweizeiliges Programm springt über GOSUB immer wieder in das Unterprogramm in Zeile 20, ohne das jedoch jeweils eine RETURN-Anweisung ausgeführt wird.

Wenn der Computer ein GOSUB durchführt, so speichert er im Stapel, wo der GOSUB-Aufruf liegt, um später wieder dorthin zurückspringen zu können.

In dem Demo-Programm füllt sich so mit jedem Unterprogramm-Aufruf der Stapel und bereits nach kurzer Zeit hat der Computer keinen Stapel-Speicherplatz mehr zur Verfügung und die OUT OF MEMORY-Fehlermeldung wird ausgegeben.

Ein derartiges Problem kann sehr schnell in komplexeren Programmen zu überraschenden Fehlerabstürzen führen. Wollen Sie sich davor schützen, so können Sie zum einen die Anti-Error Routine am Anfang dieses Buches einsetzten.

Dies ist jedoch nur ein uneleganter Notbehelf. Besser ist es, gleich sorgfältig zu planen. Von jedem GOSUB muß mittels RETURN wieder zurückgesprungen werden. Das klingt ganz einfach, doch bei umfangreichen Programmen ist es sehr schwer, den Überblick zu behalten.

Ein weiteres Problem ist, daß der Stapel-Überlauf nicht nur durch fehlerhafte Unterprogramm-Konstruktionen, sondern durch vielfältige weitere Fehlerquellen auftreten kann. Ein Beispiel dafür sind nicht abgeschlossene FOR-NEXT Schleifen: wie...

```
10 FOR A=1 TO 100
20    IF A=20 THEN 40
30 NEXT A
40 ...
```

Derartige Konstruktionen lassen alle Einträge im Stapel zurück, welche diesen mehr oder weniger schnell auffüllen und zu einem OUT OF MEMORY-Absturz führen.

Sie können jedoch mit einer kleinen Routine den Überblick über den derzeit im Stapel verfügbaren Speicherplatz behalten. Laden Sie dazu das folgende MC-Programm in den Speicher:

```
10 FOR X=50200 TO 50207
20 :.   READ Z
30 :    POKE X,Z
40 NEXT X
50 :
60 DATA 186
70 DATA 169,0
80 DATA 32,205,189
90 DATA 96
```

Das MC-Listing sieht dazu so aus:

```
tsx
lda #$00
jsr $bdcd
rts
```

Die Routine verwendet die Routine in $BBCD (48589), welche ein Low und High Byte in eine Integerzahl umwandelt und diese auf dem Bildschirm ausgibt. Diese Zahl entspricht hier dem noch freien Stapel-Speicherplatz.

Die Routine wird folgendermaßen angewendet: An allen relevanten Stellen in Ihrem Programm fügen Sie einen

```
SYS 50200:PRINT "_ZEILE XXX"
```

Befehl ein. Damit erhalten Sie eine Meldung, wieviel Raum in dem Stapel noch frei ist. Wird diese Zahl verdächtig klein, so können Sie direkt an der Meldung "Zeile (aktuelle Zeilennummer)" feststellen, wo im Programm der Stapelüberlauf statt findet.

1.18 Cursorpositionierung

Für eine komfortable Ein- und Ausgabe auf dem Bildschirm wäre es sehr nützlich, den Cursor direkt auf jede gewünschte Position des Bildschirms setzen zu können. Der Commodore 64 hat lediglich Befehle zur Positionierung des Cursors in einer Zeile, den TAB-Befehl sowie die Funktion POS zur Abfrage der Spalte. Mit dem TAB-Befehl ist zudem nur eine Positionierung nach vorne möglich.

Das Betriebssystem hält jedoch schon die Routinen zur Positionierung des Cursors bereit. In zwei Speicherstellen in der Zeropage stehen jeweils die Zeile und Spalte, in der sich der Cursor gerade befindet. Durch Abfrage dieser Werte mit PEEK können wir jederzeit die Position erfahren.

```
100 PRINT "DER CURSOR STEHT IN ZEILE " PEEK(214)
    "SPALTE" PEEK(211)
```

Wollen wir den Cursor setzen, so genügt es nicht, in Adresse 214 und 211 die entsprechenden Werte zu POKEn. Das Betriebssystem muß noch aus der Cursorposition die erforderlichen Zeiger für Bildschirm- und Farb-RAM berechnen. Dafür steht jedoch eine Routine im Betriebssytem zur Verfügung.

```
100 REM CURSOR SETZEN
110 INPUT "ZEILE ";Z
120 INPUT "SPALTE";S
130 POKE 214,Z
140 POKE 211,S
150 SYS 58640
160 PRINT "TEST";
```

Durch den Aufruf dieser Routine wird der Cursor an die gewünschte Stelle gesetzt.

Durch Kombination dieser beiden Verfahren ergeben sich neue Möglichkeiten der Programmierung. Sie können in Ihrem Programm z.B. eine Statuszeile vorsehen, in der von Fall zu Fall Anweisungen an den Bediener ausgeben werden. Um die normale Ausgabe dadurch in ihrem Verlauf nicht zu stören, merken Sie sich vorher die augenblickliche Position des Cursors und setzen dann den Cursor zur Ausgabe in die Statuszeile. Hier geben Sie nun die Meldung aus. Anschließend setzen Sie den Cursor wieder an die ursprüngliche Stelle und fahren im Programmablauf fort. Ein Programmausschnitt sieht dann etwa so aus:

```
...
300 Z=PEEK(214): REM ZEILE
310 S=PEEK(211): REM SPALTE
320 POKE 214,0: REM CURSOR IN ZEILE 0
330 POKE 211,10: REM CURSOR IN SPALTE 10
340 PRINT "BITTE DISKETTE EINLEGEN"
350 POKE 214,Z: REM ZEILE WIEDER HERSTELLEN
360 POKE 211,S: REM SPALTE
370 SYS 58640
...
```

Die Numerierung der Zeilen und Spalten geht von 0 bis 24 bzw. von 0 bis 39.

1.19 Modifizierter PRINT-Befehl

Leider ist es oft so, daß das BASIC gerade für solche Anwendungen zu langsam und unflexibel ist. Im folgenden wurde deshalb ein Assemblerprogramm aufgelistet, mit dem es möglich ist, durch einen einfachen SYS-Befehl den Befehl "PRINT" durch sehr sinnvolle Funktionen zu erweitern:

```
20:   c000                    .opt p,oo
30:   c000                    *=   49152
40:                   ;
50:                   ;***********************************
60:                   ;***                              ***
70:                   ;***    modifiziertes print       ***
80:                   ;***   --------------------       ***
90:                   ;***                              ***
100:                  ;***********************************
110:                  ;
120:  aefd            chkkom   =    $aefd
130:  b79e            getbyt   =    $b79e
140:                  ;
150:  c000 4c 1a c0           jmp  print
160:                  ;
170:                  ;*** get string ***
180:                  ;-----------------
190:                  ;
200:  c003 18         getstr   clc          ; uebertrag loeschen
210:  c004 20 9e ad           jsr  $ad9e
220:  c007 20 8f ad           jsr  $ad8f
230:  c00a 4c a6 b6           jmp  $b6a6
240:                  ;
250:                  ;*** chek komma ***
260:                  ;-----------------
270:                  ;
280:  c00d 20 79 00 komma      jsr  $0079    ; laufendes zeichen holen
290:  c010 c9 2c              cmp  #$2c     ; mit "," vergleichen
300:  c012 d0 05              bne  nokom    ; ungleich
310:  c014 20 73 00           jsr  $0073
```

```
320:  c017 a9 00          lda  #0
330:  c019 60       nokom rts
340:                ;
350:                ;********************
360:                ;*** print-befehl ***
370:                ;********************
380:                ;
390:  c01a 20 fd ae print   jsr  chkkom  ; prueft auf komma
400:  c01d 20 0d c0         jsr  komma  ; prueft auf weiteres komma
410:  c020 d0 28            bne  normprt ; kein weiteres komma
420:                ;
430:  c022 20 03 c0         jsr  getstr  ; string holen
431:  c025 8d 63 c0         sta  laenge
432:  c028 8d 63 c0         sta  laenge  ; laenge zwischenspeichern
440:  c02b a5 22            lda  $22
440:  c02d 85 fa            sta  $fa
440:  c02f a5 23            lda  $23
440:  c031 85 fb            sta  $fb     ; zeiger retten
460:  c033 20 fd ae         jsr  chkkom  ; prueft auf komma
470:  c036 20 9e b7         jsr  getbyt  ; holt anzahl wiederholungen
480:  c039 a0 00    lab1    ldy  #0
480:  c03b b1 fa    lab2    lda  ($fa),y
480:  c03d 20 d2 ff         jsr  $ffd2   ; string x mal
490:  c040 c8               iny
490:  c041 cc 63 c0         cpy  laenge
490:  c044 d0 f5            bne  lab2
500:  c046 ca               dex
500:  c047 d0 f0            bne  lab1
510:  c049 60               rts
520:                ;
530:  c04a 20 9e b7 normprt jsr  getbyt  ; x-koordinate
540:  c04d 86 d3            stx  $d3
550:  c04f 20 fd ae         jsr  chkkom
560:  c052 20 9e b7         jsr  getbyt  ; y-koordinate
560:  c055 86 d6            stx  $d6
570:  c057 20 6c e5         jsr  $e56c   ; cursor setzen
580:  c05a 20 0d c0         jsr  komma   ; folgt komma
590:  c05d d0 03            bne  fertig  ; nein
600:  c05f 4c a4 aa         jmp  $aaa4   ; sonst zum print-befehl
610:  c062 60       fertig  rts
```

Und hier wieder der BASIC-Lader, für alle Nicht- Assembler-programmierer:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***   modifiziertes print      ***
130 rem ***  ----------------------    ***
140 rem ***                            ***
150 rem ********************************
160 :
170 for x=49152 to 49250
180 :read a:poke x,a
190 :p=p+a
200 next x
210 :
220 if p<>12708 then print "fehler in datas !!!
230 :
240 end
250 :
50000 data  76, 26,192, 24, 32,158,173
50010 data  32,143,173, 76,166,182, 32
50020 data 121,  0,201, 44,208,  5, 32
50030 data 115,  0,169,  0, 96, 32,253
50040 data 174, 32, 13,192,208, 40, 32
50050 data   3,192,141, 99,192,141, 99
50060 data 192,165, 34,133,250,165, 35
50070 data 133,251, 32,253,174, 32,158
50080 data 183,160,  0,177,250, 32,210
50090 data 255,200,204, 99,192,208,245
50100 data 202,208,240, 96, 32,158,183
50110 data 134,211, 32,253,174, 32,158
50120 data 183,134,214, 32,108,229, 32
50130 data  13,192,208,  3, 76,164,170
50140 data  96
```

Doch nun zur genauen Anwendung. Dieser SYS-Aufruf beinhaltet eigentlich zwei Funktionen. Mit der ersten ist es möglich, den Cursor an eine bestimmte Stelle zu positionieren und dann wahlweise noch einen Text auszugeben. Der Aufruf hierfür ist:

```
SYS 49152,XPOS,YPOS (,"text"(;))
```

oder

```
SYS 49152,XPOS,YPOS (,wert(;))
```

Wobei "text" ein String ist und 'wert' ein beliebiger Ausdruck.

Die zweite Möglichkeit, die diese Erweiterung bietet, ist folgende:

```
SYS 49152,,"string",wiederholung
```

Bei diesem Aufruf wird der String "string" so oft nach der aktuellen Cursorposition ausgegeben, wie es der Wert "wiederholung" besagt. Beachten Sie dabei zwei Dinge:

1. Nach dem Aufruf des Assemblerprogramms müssen unbedingt zwei Kommas folgen.

2. Der Wert 'wiederholung' darf den Wert 255 nicht überschreiten.

1.20 Eine komfortable INPUT-Routine

Die C-64 INPUT Funktion ist und bleibt ein unbefriedigendes Behelfsmittel. Zur komfortablen und fehlerfreien Programmbedienung ist sie nicht geeignet.

Da ist zunächst einmal das antiquarisch, hoffnungslos veraltet wirkende Fragezeichen. Dieses kann man schnell unterdrücken, indem man entweder GET einsetzt, was mit größerem Aufwand verbunden ist, oder indem man Input als File eröffnet.

Dies sieht dann beispielsweise so aus:

```
10 OPEN 1,0:INPUT#1, T$: PRINT T$: CLOSE 1
```

Wir müssen uns dann aber noch mit dem Problem der "unerlaubten" Eingaben herumschlagen. Zeichen wie Komma, Doppelpunkt oder Semikolon werden nicht akzeptiert.

Hier kann man jedoch auf eine sehr elegante Art und Weise Abhilfe schaffen. Verwenden Sie dazu die folgende Routine:

```
10 SYS 42336: EL=40
20 FOR I=512 TO 512+EL
30 IF PEEK(I) THEN T$=T$+CHR$(PEEK(I)): NEXT I
40 PRINT T$
```

Die Variable EL legt hierbei die Eingabenlänge fest. Die Systemroutine beginnend mit der Adresse 42336 kopiert alle Zeichen einer Bildschirmzeile in den Tastaturbuffer.

Die Variable T$ liest nun in der FOR-Schleife alle Zeichen des Buffers, bis ein Nullwert das Eingabeende anzeigt, PEEK(I) also 0 ist.

Besonders bestechend an dieser Routine ist ihre Kompaktheit.

Es geht auch noch komfortabler und effektiver. Das nun folgende INPUT-Unterprogramm ist dafür aber auch etwas umfangreicher. Aber der Aufwand lohnt sich:

```
5 PRINT"□": POKE 53281,0
10 X=4:Y=8:GOSUB 62000:MX=5:PRINT "EINGABE
..> ";:GOSUB 60000
20 PRINT:PRINT"■"U$
30 STOP
40 :
50 :
60000 REM EINGABEROUTINE
60002 L$="":R$=" "
60005 DI=15:MX=6
60010 CS=0:ZT=TI+DI
60020 IF ZT<TI THEN CS=NOT CS:ZT=TI+DI
60030 :    T$=LEFT$(R$,1)+"■"
60040 :    PRINT MID$("■■",CS+2,1);T$;
60050 GET K$:IF K$="" THEN 60020
60060 :    GOSUB 61000:REM BEEP
60070 :       IFK$=CHR$(13)THENPRINT"■"R$:U$
=LEFT$(L$+R$,LEN(L$+R$)-1):RETURN
60080 :       IF ASC(K$)=34 THEN K$="'":REM
HOCHKOMMA
60090 :       IF(ASC(K$) AND 127)<31 THEN 60
500
60100 :       IF LEN(L$+R$)>MX AND LEN(R$)=1
 THEN 60020
60120 : L$=L$+K$: IF R$<>" "THEN R$=MID$(R
$,2)
60130 PRINT"■"K$;:GOTO 60010
60200 :
60500 H$="■"+T$+K$
60520 IFASC(K$)=148ANDLEN(L$+R$)<MX+1 THEN
 R$=" "+R$:PRINT "■"T$"■ ■";
60530 IF ASC(K$)=20 AND LEN(L$)>0 THEN L$=
LEFT$(L$,LEN(L$)-1):PRINT H$;
60540 IFK$="■"ANDLEN(L$)>0THENR$=RIGHT$(L$
,1)+R$:L$=LEFT$(L$,LEN(L$)-1):PRINTH$;
60550 IFK$="■"ANDLEN(R$)>1THENL$=L$+LEFT$(
```

```
R$,1):R$=RIGHT$(R$,LEN(R$)-1):PRINTH$;
60570 GOTO 60020
60990 :
61000 REM TASTATURBEEP
61010 XX=54272: HOEHE=50
61020 POKE XX+24,15: POKE XX+6, 240
61030 POKE XX+4, 33: POKE XX+23,4
61040 POKE 30,HOEHE
61050 POKE XX+1,HOEHE
61100 FOR  X=1 TO 100:NEXT X
61110 POKE XX+24,0
61120 RETURN
61900 :
62000 REM X -  Y KOORDINATEN
62010 POKE 214,Y:POKE 211,Y
62020 SYS 58640
62030 RETURN
```

Diese Routine bietet Ihnen ein Optimum an Komfort:

Probieren Sie einmal die Demo-Eingabe in den Zeile 10 und 20 vor der Routine aus.

Alle Zeichen werden akzeptiert. Mit den Parametern x und y können Sie den Input-Text beliebig auf dem Bildschirm positionieren.

Die Inputlänge wird über die Variable MX festgelegt. Dadurch können Sie ein Eingabefeld von einem bis maximal 255 Zeichen festlegen.

Ferner läßt sich die Schrift-Farbe festlegen, wie in dem Demo vorgeführt wurde.

Eine große Stärke ist die universelle Editiermöglichkeit bei der Eingabe. Cursor-left, Cursor-right, Insert und Delete sind realisiert. Alle anderen Editiertasten, wie Cursor-up oder Cursor-down, sind sinnvollerweise gesperrt.

Die Frequenz des Cursorblinkens läßt sich über die Variable DI in der Zeile 60005 frei wählen.

Zudem enthält unsere Routine einen Tastaturpieps-Befehl. Die Tonhöhe können Sie in der Zeile 61010 über die Variable HOEHE beliebig bestimmen. Die Tonlänge wird durch die Scleife in 61100 festgelegt. Der Vorteil dieses variablen Piepsers ist, daß hier zum einen beliebige Tonänderungen möglich sind.

Sie können also für verschiedene Eingaben unterschiedliche Piepser festlegen.

Zum anderen wird im Gegensatz zu normalen, herkömmlichen Tastaturpiepsern die Interruproutine nicht belegt. Sie bleibt also frei für andere Anwendungen, wie beispielsweise die ERROR-Routine, auf die wir später in diesem Kapitel noch kommen.

Übrigens ist unsere Routine nicht nur auf die Eingabe von Texten beschränkt, sondern läßt sich auch für Zahlen universell einsetzen. Dazu können Sie den val-Wert der Variable u$ bestimmen, der dem eingegebenen Zahlenwert entspricht.

Hier gibt es zusätzlich die Möglichkeit, die Eingabe nur auf Zahlenwerte abzustimmen, Buchstaben werden dabei nicht akzeptiert.

Dies geht folgendermaßen vor sich: Fügen Sie zunächst die folgende Erweiterung zu der Routine an:

```
62040 :
63000 IF MK=4 THEN RETURN
63010 IF U$="" THEN 63030
63020 IF U$="0" THEN U=0:RETURN
63030 U=VAL(U$)
63040 IF U<>0 THEN RETURN
63050 FOR XX=0 TO LEN(U$):PRINT"■■";:NEXT X
X
```

```
63060 FOR XX=0 TO LEN(U$):PRINT" ";:NEXT X
X
63070 FOR XX=0 TO LEN(U$):PRINT"■■";:NEXT X
X:GOTO 60000
```

Nun ändern Sie den RETURN-Befehl in Zeile 60070 in ein

```
GOTO 63000
```

Ob die Eingabe eine Zahl oder ein Text sein soll, bestimmen Sie über den Wert der Variablen MK beim Aufruf der Routine.

Ist der Wert von MK = 4, dann erwartet der Computer eine Texteingabe. Ein Demo-Aufruf sieht beispielsweise folgendermaßen aus:

```
10 MK=4: PRINT "TEXTEINGABE ...> ";:MX=10:GOSUB 60000
20 PRINT U$
```

Die Eingabe einer maximal 6 stelligen Zahl wird dagegen folgendermaßen realisiert:

```
10 MK=0: PRINT "WERT = ";:MX=8: GOSUB 60000
20 PRINT U
```

Die Variable U enthält dann den Wert der eigegebenen Zahl. Haben Sie jedoch in diesem Beispiel keine Zahl, sondern beispielsweise Buchstaben eingegeben, so erkennt die erweiterte Routine dies. Die fehlerhafte Eingabe wird automatisch in den Zeilen 63050ff gelöscht und der Computer erwartet nun eine korrekte Zahl.

Der große Vorteil dabei liegt darin, daß dabei der INPUT-Text nicht gelöscht wird und dadurch die Bildschirmmaske vollständig erhalten bleibt.

Übrigens werden natürlich auch Kommazahlen akzeptiert.

Experimentieren Sie mit der erweiterten Routine. Ich bin sicher, daß Sie diese universelle und superkomfortable INPUT-Routine optimal in Ihre Programme integrieren können und diesen dadurch einen "Professionellen Toutch" geben.

Input-Routine in Assembler

Sicher standen Sie auch schon einmal vor dem Problem, daß Ihr Programm in dem Sie eine Tastaturabfrage mit dem normalen INPUT-Befehl durchführten Ihre Bildschirmmaske zerstörte, weil irgendwelche Steuerzeichen, z.B. "CLR", darin enthalten waren. Oder aber Sie standen vor dem Problem, daß Sie mehrere Eingabefelder innerhalb einer Maske hatten, die Sie bei Bedarf anspringen mußten.

Nun mußte aber eine eigene Eingaberoutine her. Wie aber? - In BASIC ist dieses wohl möglich, aber oft viel zu langsam. Deswegen stellen wir im folgenden eine komfortable INPUT-Routine an, mit der selbst Eingabefelder in Routinemasken kein Problem mehr darstellen sollten.

Zur Funktionsweise das Assemblerprogramms, das ab der Adresse 49152 ($C000) liegt: Der Aufruf der F-Input-Routine lautet:

```
SYS 49152,laengestr,xk,yk,sting$
```

Dabei werden folgende Parameter verwendet:

```
laengestr  - die maximale Anzahl von Buchstaben, die
             eingegeben werden dürfen.
xk         - die X-Koordinate (0-39) des Eingabefeldes.
yk         - die Y-Koordinate (0-24) des Eingabefeldes.
string$    - eine Variable des Types STRING, in die der
             Eingabetext gespeichert wird.
```

Für alle Interessierten folgt nun das Assemblerlisting, direkt danach der BASIC-Lader:

```
110:  c000                    .opt p,oo
110:  c000                    .sym 4
120:  c000                    *=   49152
130:                  ;
140:                  ;*** formatierter input befehl ***
150:                  ;
160:                  ;a = laenge des strings
170:                  ;x = x-position
180:                  ;y = y-position
190:                  ;
200:  c000 4c 08 c0           jmp  forminp
210:                  ;
220:  c003 05         curcol  .byt 5
230:  c004 00         laengez .byt 0
240:  c005 00         charuc  .byt 0
250:  c006 00         coluc   .byt 0
260:  c007 00         xz      .byt 0
270:  0064            vorbes  =    100
280:                  ;
290:  c008 20 fd ae forminp   jsr  $aefd
290:  c00b 20 9e b7           jsr  $b79e
290:  c00e 8e 04 c0           stx  laengez
300:  c011 20 fd ae           jsr  $aefd
300:  c014 20 9e b7           jsr  $b79e
300:  c017 8e 07 c0           stx  xz
310:  c01a 20 fd ae           jsr  $aefd
310:  c01d 20 9e b7           jsr  $b79e
310:  c020 8a                 txa
310:  c021 a8                 tay
320:                  ;
330:  c022 a9 00              lda  #<1024
330:  c024 ae 88 02           ldx  648
340:  c027 85 fa              sta  $fa
340:  c029 86 fb              stx  $fb
350:                  ;
360:  c02b a5 fa              lda  $fa
```

```
360:   c02d 18               clc
360:   c02e 6d 07 c0         adc  xz
360:   c031 85 fa            sta  $fa
370:   c033 90 02            bcc  ok0
370:   c035 e6 fb            inc  $fb
380:                ;
390:   c037 4c 45 c0 ok0      jmp  ok2
400:   c03a a5 fa    ok1     lda  $fa
400:   c03c 18               clc
400:   c03d 69 28            adc  #40
400:   c03f 85 fa            sta  $fa
410:   c041 90 02            bcc  ok2
410:   c043 e6 fb            inc  $fb
420:   c045 88       ok2     dey
420:   c046 10 f2            bpl  ok1
420:   c048 a5 fa            lda  $fa
420:   c04a 85 fc            sta  $fc
420:   c04c a5 fb            lda  $fb
420:   c04e 29 03            and  #$03
420:   c050 09 d8            ora  #$d8
420:   c052 85 fd            sta  $fd
430:   c054 ac 04 c0         ldy  laengez
430:   c057 88               dey
430:   c058 8c 04 c0         sty  laengez
430:   c05b a9 64    vbla    lda  #vorbes
430:   c05d 91 fa            sta  ($fa),y
430:   c05f ad 86 02         lda  646
440:   c062 91 fc            sta  ($fc),y
440:   c064 88               dey
440:   c065 10 f4            bpl  vbla
440:   c067 a0 00            ldy  #0
450:                ;
460:                ;*** eingabe schleife ***
470:                ;
480:                ;** bildschirm-code -> farbecode
490:                ;
500:   c069 a5 fa    hauptlab lda  $fa
500:   c06b 85 fc            sta  $fc
510:   c06d a5 fb            lda  $fb
510:   c06f 29 03            and  #$03
```

```
510:   c071 09 d8            ora  #$d8
520:   c073 85 fd            sta  $fd
530:                  ;
540:   c075 b1 fa            lda  ($fa),y
540:   c077 8d 05 c0         sta  charuc
550:   c07a 49 80            eor  #$80
550:   c07c 91 fa            sta  ($fa),y
560:   c07e b1 fc            lda  ($fc),y
560:   c080 8d 06 c0         sta  coluc
570:   c083 ad 03 c0         lda  curcol
570:   c086 91 fc            sta  ($fc),y
580:                  ;
590:   c088 8a        stin    txa
590:   c089 48               pha
590:   c08a 98               tya
590:   c08b 48               pha
600:                  ;
610:   c08c 20 e4 ff einlab2  jsr  $ffe4
610:   c08f f0 fb            beq  einlab2
620:   c091 85 02            sta  2
620:   c093 68               pla
620:   c094 a8               tay
620:   c095 68               pla
620:   c096 aa               tax
620:   c097 a5 02            lda  2
630:   c099 a2 03            ldx  #3
640:   c09b dd a6 c0 einlab3 cmp  chartab,x
650:   c09e f0 ec            beq  einlab2
660:   c0a0 ca               dex
670:   c0a1 10 f8            bpl  einlab3
680:                  ;
690:   c0a3 4c aa c0         jmp  finput2
700:                  ;
710:   c0a6 5f 5e 24 chartab  .byt 95,94,36,160
720:                  ;
730:   c0aa c9 0d    finput2  cmp  #13
730:   c0ac f0 33            beq  end1
740:   c0ae c9 11            cmp  #17
740:   c0b0 f0 2f            beq  end1
750:   c0b2 c9 91            cmp  #145
```

```
750:   c0b4 f0 2b            beq  end1
760:   c0b6 c9 85            cmp  #133
760:   c0b8 f0 27            beq  end1
770:   c0ba c9 89            cmp  #137
770:   c0bc f0 23            beq  end1
780:   c0be c9 8d            cmp  #141
780:   c0c0 f0 1f            beq  end1
790:                ;
800:                ;
810:   c0c2 c9 1d            cmp  #29
810:   c0c4 f0 20            beq  curright
820:   c0c6 c9 9d            cmp  #157
820:   c0c8 f0 1f            beq  curleft
830:   c0ca c9 86            cmp  #134
830:   c0cc f0 1e            beq  clrinput ;clr eingabe
840:   c0ce c9 87            cmp  #135
840:   c0d0 f0 1d            beq  beginp  ;anfang eingabe
850:   c0d2 c9 88            cmp  #136
850:   c0d4 f0 1c            beq  endinp  ;ende eingabe
860:   c0d6 c9 14            cmp  #20
860:   c0d8 f0 1b            beq  del     ;zeichen loeschen
870:   c0da c9 94            cmp  #148
870:   c0dc f0 1a            beq  ins     ;zeichen einfuegen
880:                ;
890:   c0de 4c fb c0          jmp  normchar
900:   c0e1 85 f9    end1    sta  $f9
900:   c0e3 4c d5 c1          jmp  endend
910:   c0e6 4c 49 c1 curright jmp  curr2
920:   c0e9 4c 3e c1 curleft  jmp  curleft2
930:   c0ec 4c 60 c1 clrinput jmp  clrin2
940:   c0ef 4c 4f c1 beginp   jmp  beginp2
950:   c0f2 4c 57 c1 endinp   jmp  endinp2
960:   c0f5 4c 7e c1 del      jmp  del2
970:   c0f8 4c a3 c1 ins      jmp  ins2
980:                ;
990:                ;*** pruefung auf erlaubtes zeichen
1000:               ;
1010:  c0fb c9 20    normchar cmp  #032
1010:  c0fd 90 18             bcc  steuer
1020:  c0ff c9 40             cmp  #064
```

```
1020:  c101 90 26            bcc  endnc
1030:  c103 c9 60            cmp  #096
1030:  c105 90 13            bcc  min64
1040:  c107 c9 80            cmp  #128
1040:  c109 90 15            bcc  min32
1050:  c10b c9 a0            cmp  #160
1050:  c10d 90 08            bcc  steuer
1060:  c10f c9 c0            cmp  #192
1060:  c111 90 07            bcc  min64
1070:  c113 c9 ff            cmp  #255
1070:  c115 90 0f            bcc  min128
1080:                ;
1090:  c117 4c 88 c0 steuer  jmp  stin
1100:                ;
1110:  c11a 38       min64   sec
1110:  c11b e9 40            sbc  #64
1110:  c11d 4c 29 c1         jmp  endnc
1120:  c120 38       min32   sec
1120:  c121 e9 20            sbc  #32
1120:  c123 4c 29 c1         jmp  endnc
1130:  c126 38       min128  sec
1130:  c127 e9 80            sbc  #128
1140:                ;
1150:  c129 20 c8 c1 endnc   jsr  charback
1150:  c12c 91 fa            sta  ($fa),y
1150:  c12e 48               pha
1150:  c12f ad 86 02         lda  646
1150:  c132 91 fc            sta  ($fc),y
1150:  c134 68               pla
1160:  c135 cc 04 c0 currr   cpy  laengez
1160:  c138 b0 01            bcs  zulang
1170:  c13a c8               iny
1180:  c13b 4c 69 c0 zulang  jmp  hauptlab
1190:                ;
1200:  c13e 20 c8 c1 curleft2 jsr  charback
1210:  c141 c0 00            cpy  #0
1210:  c143 f0 f6            beq  zulang
1220:  c145 88               dey
1220:  c146 4c 69 c0         jmp  hauptlab
1230:                ;
```

```
1240:  c149 20 c8 c1 curr2    jsr  charback
1240:  c14c 4c 35 c1          jmp  currr
1250:                ;
1260:  c14f 20 c8 c1 beginp2  jsr  charback
1270:  c152 a0 00             ldy  #0
1270:  c154 4c 69 c0          jmp  hauptlab
1280:                ;
1290:  c157 20 c8 c1 endinp2  jsr  charback
1300:  c15a ac 04 c0          ldy  laengez
1300:  c15d 4c 69 c0          jmp  hauptlab
1310:                ;
1320:  c160 20 c8 c1 clrin2   jsr  charback
1320:  c163 a5 02             lda  2
1320:  c165 48                pha
1320:  c166 88                dey
1320:  c167 84 02             sty  2
1330:  c169 ac 04 c0          ldy  laengez
1340:  c16c a9 64             lda  #vorbes
1350:  c16e 91 fa    clrlab   sta  ($fa),y
1350:  c170 88                dey
1350:  c171 c4 02             cpy  2
1350:  c173 d0 f9             bne  clrlab
1350:  c175 a4 02             ldy  2
1350:  c177 c8                iny
1360:  c178 68                pla
1360:  c179 85 02             sta  2
1360:  c17b 4c 69 c0          jmp  hauptlab
1370:                ;
1380:  c17e 20 c8 c1 del2     jsr  charback
1380:  c181 cc 04 c0          cpy  laengez
1380:  c184 f0 55             beq  ins3
1390:  c186 a5 02             lda  2
1390:  c188 48                pha
1400:  c189 84 02             sty  2
1410:                ;
1420:  c18b c8        dellab  iny
1420:  c18c b1 fa             lda  ($fa),y
1420:  c18e 88                dey
1420:  c18f 91 fa             sta  ($fa),y
1420:  c191 c8                iny
```

```
1430:  c192 cc 04 c0          cpy  laengez
1430:  c195 d0 f4             bne  dellab
1440:                  ;
1450:  c197 a9 64             lda  #vorbes
1450:  c199 91 fa             sta  ($fa),y
1460:  c19b a4 02             ldy  2
1460:  c19d 68                pla
1460:  c19e 85 02             sta  2
1460:  c1a0 4c 69 c0          jmp  hauptlab
1470:                  ;
1480:  c1a3 20 c8 c1 ins2     jsr  charback
1480:  c1a6 cc 04 c0          cpy  laengez
1480:  c1a9 f0 30             beq  ins3
1490:  c1ab a5 02             lda  2
1490:  c1ad 48                pha
1500:  c1ae 84 02             sty  2
1500:  c1b0 ac 04 c0          ldy  laengez
1510:  c1b3 88       inslab   dey
1510:  c1b4 b1 fa             lda  ($fa),y
1510:  c1b6 c8                iny
1510:  c1b7 91 fa             sta  ($fa),y
1520:  c1b9 88                dey
1520:  c1ba c4 02             cpy  2
1520:  c1bc d0 f5             bne  inslab
1530:  c1be a9 20             lda  #32
1530:  c1c0 91 fa             sta  ($fa),y
1540:  c1c2 68                pla
1540:  c1c3 85 02             sta  2
1540:  c1c5 4c 69 c0          jmp  hauptlab
1550:                  ;
1560:                  ;
1570:  c1c8 48       charback pha
1570:  c1c9 ad 05 c0          lda  charuc
1570:  c1cc 91 fa             sta  ($fa),y
1580:  c1ce ad 06 c0          lda  coluc
1580:  c1d1 91 fc             sta  ($fc),y
1590:  c1d3 68                pla
1590:  c1d4 60                rts
1600:                  ;
1610:  c1d5 20 c8 c1 endend   jsr  charback
```

```
1610:  c1d8 4c e2 c1          jmp  endsle
1620:                ;
1630:  c1db a9 64     ins3    lda  #vorbes
1630:  c1dd 91 fa             sta  ($fa),y
1630:  c1df 4c 69 c0          jmp  hauptlab
1640:                ;
1650:  c1e2 ac 04 c0 endsle   ldy  laengez
1660:  c1e5 b1 fa     slelab  lda  ($fa),y
1670:  c1e7 c9 64             cmp  #vorbes
1680:  c1e9 d0 09             bne  ueber
1690:  c1eb 88                dey
1700:  c1ec 10 f7             bpl  slelab
1700:  c1ee 20 fd ae          jsr  $aefd
1700:  c1f1 4c 8b b0          jmp  $b08b
1710:                ;
1720:  c1f4 c8        ueber   iny
1720:  c1f5 8c 35 c2          sty  laengstr
1720:  c1f8 b1 fa     lsflab  lda  ($fa),y
1720:  c1fa c9 64             cmp  #vorbes
1720:  c1fc d0 02             bne  uw1
1720:  c1fe a9 20             lda  #32
1730:  c200 85 d7     uw1     sta  $d7
1730:  c202 29 3f             and  #$3f
1730:  c204 06 d7             asl  $d7
1730:  c206 24 d7             bit  $d7
1730:  c208 10 02             bpl  wskip1
1730:  c20a 09 80             ora  #$80
1730:  c20c 90 04     wskip1  bcc  wskip2
1740:  c20e a6 d4             ldx  $d4
1740:  c210 d0 04             bne  wskip3
1740:  c212 70 02     wskip2  bvs  wskip3
1740:  c214 09 40             ora  #$40
1750:  c216 99 38 c2 wskip3   sta  buffer,y
1760:  c219 88                dey
1770:  c21a 10 dc             bpl  lsflab
1780:                ;
1790:  c21c 20 fd ae          jsr  $aefd
1800:  c21f 20 8b b0          jsr  $b08b
1810:  c222 a5 47             lda  $47
1810:  c224 85 49             sta  $49
```

```
1810:  c226 a5 48            lda  $48
1810:  c228 85 4a            sta  $4a
1820:  c22a a9 35            lda  #<laengstr
1820:  c22c a0 c2            ldy  #>laengstr
1830:  c22e 85 64            sta  $64
1830:  c230 84 65            sty  $65
1840:  c232 4c 52 aa         jmp  $aa52
1850:                ;
1860:  c235 00 38 c2 laengstr .byt 0,<buffer,>buffer
1870:  c238          buffer  =    *
```

symboltable:

```
buffer    c238   laengstr  c235   wskip3    c216   wskip2    c212
wskip1    c20c   uw1       c200   lsflab    c1f8   ueber     c1f4
slelab    c1e5   endsle    c1e2   ins3      c1db   endend    c1d5
charback  c1c8   inslab    c1b3   ins2      c1a3   dellab    c18b
del2      c17e   clrlab    c16e   clrin2    c160   endinp2   c157
beginp2   c14f   curr2     c149   curleft2  c13e   zulang    c13b
currr     c135   endnc     c129   min128    c126   min32     c120
min64     c11a   steuer    c117   normchar  c0fb   ins       c0f8
del       c0f5   endinp    c0f2   beginp    c0ef   clrinput  c0ec
curleft   c0e9   curright  c0e6   end1      c0e1   finput2   c0aa
chartab   c0a6   einlab3   c09b   einlab2   c08c   stin      c088
hauptlab  c069   vbla      c05b   ok2       c045   ok1       c03a
ok0       c037   forminp   c008   vorbes    0064   xz        c007
coluc     c006   charuc    c005   laengez   c004   curcol    c003

56 symbols defined
```

Und nun der BASIC-Loader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***   formatierter input       ***
130 rem ***  ---------------------     ***
140 rem ***                            ***
150 rem ********************************
160 :
170 for x=49152 to 49725
180 :read a:poke x,a
190 :p=p+a
200 next
210 :
220 if p<> 76297 then print "fehler in datas !!!":stop
230 :
240 end
250 :
50000 data  76,  8,192,  5,  0,  0,  0
50010 data   0, 32,253,174, 32,158,183
50020 data 142,  4,192, 32,253,174, 32
50030 data 158,183,142,  7,192, 32,253
50040 data 174, 32,158,183,138,168,169
50050 data   0,174,136,  2,133,250,134
50060 data 251,165,250, 24,109,  7,192
50070 data 133,250,144,  2,230,251, 76
50080 data  69,192,165,250, 24,105, 40
50090 data 133,250,144,  2,230,251,136
50100 data  16,242,165,250,133,252,165
50110 data 251, 41,  3,  9,216,133,253
50120 data 172,  4,192,136,140,  4,192
50130 data 169,100,145,250,173,134,  2
50140 data 145,252,136, 16,244,160,  0
50150 data 165,250,133,252,165,251, 41
50160 data   3,  9,216,133,253,177,250
50170 data 141,  5,192, 73,128,145,250
50180 data 177,252,141,  6,192,173,  3
50190 data 192,145,252,138, 72,152, 72
50200 data  32,228,255,240,251,133,  2
```

```
50210 data 104,168,104,170,165,  2,162
50220 data   3,221,166,192,240,236,202
50230 data  16,248, 76,170,192, 95, 94
50240 data  36,160,201, 13,240, 51,201
50250 data  17,240, 47,201,145,240, 43
50260 data 201,133,240, 39,201,137,240
50270 data  35,201,141,240, 31,201, 29
50280 data 240, 32,201,157,240, 31,201
50290 data 134,240, 30,201,135,240, 29
50300 data 201,136,240, 28,201, 20,240
50310 data  27,201,148,240, 26, 76,251
50320 data 192,133,249, 76,213,193, 76
50330 data  73,193, 76, 62,193, 76, 96
50340 data 193, 76, 79,193, 76, 87,193
50350 data  76,126,193, 76,163,193,201
50360 data  32,144, 24,201, 64,144, 38
50370 data 201, 96,144, 19,201,128,144
50380 data  21,201,160,144,  8,201,192
50390 data 144,  7,201,255,144, 15, 76
50400 data 136,192, 56,233, 64, 76, 41
50410 data 193, 56,233, 32, 76, 41,193
50420 data  56,233,128, 32,200,193,145
50430 data 250, 72,173,134,  2,145,252
50440 data 104,204,  4,192,176,  1,200
50450 data  76,105,192, 32,200,193,192
50460 data   0,240,246,136, 76,105,192
50470 data  32,200,193, 76, 53,193, 32
50480 data 200,193,160,  0, 76,105,192
50490 data  32,200,193,172,  4,192, 76
50500 data 105,192, 32,200,193,165,  2
50510 data  72,136,132,  2,172,  4,192
50520 data 169,100,145,250,136,196,  2
50530 data 208,249,164,  2,200,104,133
50540 data   2, 76,105,192, 32,200,193
50550 data 204,  4,192,240, 85,165,  2
50560 data  72,132,  2,200,177,250,136
50570 data 145,250,200,204,  4,192,208
50580 data 244,169,100,145,250,164,  2
50590 data 104,133,  2, 76,105,192, 32
50600 data 200,193,204,  4,192,240, 48
```

```
50610 data 165,  2, 72,132,  2,172,  4
50620 data 192,136,177,250,200,145,250
50630 data 136,196,  2,208,245,169, 32
50640 data 145,250,104,133,  2, 76,105
50650 data 192, 72,173,  5,192,145,250
50660 data 173,  6,192,145,252,104, 96
50670 data  32,200,193, 76,226,193,169
50680 data 100,145,250, 76,105,192,172
50690 data   4,192,177,250,201,100,208
50700 data   9,136, 16,247, 32,253,174
50710 data  76,139,176,200,140, 53,194
50720 data 177,250,201,100,208,  2,169
50730 data  32,133,215, 41, 63,  6,215
50740 data  36,215, 16,  2,  9,128,144
50750 data   4,166,212,208,  4,112,  2
50760 data   9, 64,153, 56,194,136, 16
50770 data 220, 32,253,174, 32,139,176
50780 data 165, 71,133, 73,165, 72,133
50790 data  74,169, 53,160,194,133,100
50800 data 132,101, 76, 82,170,  0, 56
50810 data 194,203, 76, 65, 85, 83, 32
```

Wie Sie vielleicht schon aus dem Assemblerprogramm ersehen konnten, besitzt diese F-INPUT Routine noch einige andere Funktionen, die nicht mit im SYS-Aufruf enthalten sind.

Zunächst können Sie die Farbe des Cursors und des Eingabefeldes bestimmen. Das geschieht entweder durch Änderung von Zeile 220 im Assemblerlisting oder durch folgenden POKE:

```
POKE 49155,farbe
```

Dieser Poke muß natürlich nach dem Einladen durch den BASIC-Lader erfolgt sein.

Alle folgenden Tasten führen zum Verlassen des Eingabefeldes:

```
F1
F2
crsr down
crsr up
return
SHIFT & return
```

Damit Sie wissen, mit welcher Taste das Eingabefeld verlassen wurde - um dann z.B. in ein vorheriges oder nachfolgendes zu springen - wird der entsprechende ASCII-Code in der Speicherstelle 249 gespeichert. Die Speicherstelle können Sie mit PEEK auslesen.

Folgende weitere Funktionen können Ihnen das Editieren eines Textes innerhalb eines Eingabefeldes erleichtern:

1) F3 Durch einen Tastendruck auf die F3-Taste wird der rechts neben dem Cursor stehende Text gelöscht, also mit dem Vorbesetztzeichen (BC 100) aufgefüllt.

2) inst Diese Taste bewirkt, daß der Text neben dem Cursor nach rechts gescrollt wird und unter dem Cursor ein Leerzeichen erscheint.

3) del Durch die 'del' Taste wird das Zeichen unter dem Cursor gelöscht und der rechts neben dem Cursor stehende Text um ein Zeichen nach links gescrollt.

4) F5 Ein Druck auf die F5-Taste bewirkt, daß der Cursor an den Anfang des Eingabefeldes springt.

5) F7 Diese Funktion setzt den Cursor auf das Ende des Eingabefeldes.

1.21 Errechnen beliebiger Formeln

Haben Sie schon einmal versucht, ein wirklich universelles Rechenprogramm zu schreiben?

Unter universell verstehen wir hier ein Programm, mit dem sich beliebige arithmetische Operationen mit beliebigen Variablen oder Konstanten durchführen lassen. Die Betonung liegt auf "beliebig".

Natürlich nicht, werden Sie antworten, denn das Wesen eines Programmes liegt ja darin, nach vorher festgelegten Algorithmen mit den Daten zu verfahren.

Stellen Sie sich einmal vor, Sie wollen eine rechnende Textverarbeitung (sowas gibt es) schreiben.

In Ihrem Text haben Sie numerische Felder, die arithmetisch miteinander verknüpft werden sollen. Das ist z.B. beim Schreiben einer Rechnung der Fall, wo die Preisfelder der Einzelposten addiert werden sollen, die Summe anschließend mit dem Mehrwertsteuersatz multipliziert wird und auch noch Verpakkung und Porto hinzukommen.

Sie können selbstverständlich eine Version des oben skizzierten Programms schreiben, die speziell auf die Bedürfnisse des Rechnungschreibens abgestimmt ist.

Ein solches Programm ist sicher recht nützlich, wenn Sie die Preise der einzelnen Posten kennen und entsprechend eingeben können.

Ein Teppichhändler jedoch, der Auslegeware von der Rolle verkauft, tut sich mit diesem Programm schon wesentlich schwerer, da der Preis seiner Ware von der Quadratmeterzahl abhängt. Diese muß er zunächst berechnen und hat erst nach der Multiplikation mit dem Quadratmeterpreis den Wert, den er dem beschriebenen Rechnungsprogramm zur Verfügung stellen kann.

Kein Problem, werden Sie sagen, schreiben wir eben eine weitere Version für den Teppichhändler.

Wieviele Versionen für wieviele Händler sollen denn dann entstehen?

Wäre es nicht viel praktischer, eine einzige Version zu haben, mit der sämtliche anfallenden Berechnungsarten abgedeckt sind?

Genau das wollen wir Ihnen hier vermitteln, nämlich ein Verfahren, mit dem Sie nachträglich, d.h. während des Programmlaufes festlegen können, welche Variablen wie miteinander verknüpft werden sollen.

Das ist nur möglich, wenn es gelingt, eine BASIC-Zeile mit der gewünschten Formel beliebig oft bei laufendem Programm zu generieren.

Wie das gemacht wird, wollen wir Ihnen jetzt zeigen. Das im folgenden besprochene Programm enthält auch einen Maschinenspracheteil, aber der wird per BASIC gehandhabt.

Bevor wir näher auf die einzelnen Programmschritte eingehen, sollen Ihnen zunächst die verwendeten Variablen und Speicheradressen vorgestellt werden.

Als erstes die Variablen:

TM	Adresse des Speicherendes
VL	niederwertiges Byte der Adresse 'Variablenstart'
VH	wie oben, jedoch höherwertiges Byte
VT	wie oben, aber Gesamtwert
BU	Adresse aus dem Bereich des Zeileneingabepuffers
BC	Laufvariable zum Füllen des Puffers
CA$	eingegebene Rechenvorschrift
RE	hierin steht nach Durchlauf der Routine das Ergebnis

Die benutzten Speicherstellen:

45-46	Zeiger auf Beginn der Variablen
47-48	Zeiger auf Start der Arrays
49-50	Zeiger auf Beginn der Strings
56	höherwertiges Byte des Zeigers auf Ende des BASIC-RAM
40448	hier steht die erzeugte BASIC-Zeile 50100
40704	ab hier steht die Routine, die aus dem Inhalt des Eingabepuffers eine BASIC-Zeile erzeugt und nach 40448 bringt

Für diejenigen, die es interessiert, das Maschinenprogramm:

```
        lda $7a                 ; BASIC-Pointer retten
        sta $9fff
        lda $7b
        sta $9ffe
        lda $14
        sta $9ffd
        lda $15
        sta $9ffc
        lda #$0b                ; Offset auf INPUT-Buffer
        sta $7a
        jsr $a579               ; Aufruf der Routine 'CRUNCH'
        ldx #0
        lda $0200,x             ; Zeile nach 40453 transferieren
        beq  YY                 ; Aussprung wenn Ende
        sta $9e05,x
        inx
        bne XX
YY      lda #$3a                ; hinter die Zeile
        sta $9e05,x
        lda #$8e                ; RETURN anhängen
        sta $9e06,x
        lda #0                  ; Zeilenende markieren
        sta $9e07,x
        sta $9e08,x
```

```
sta $9e09,x
lda $9fff                ; BASIC-Pointer zurückladen
sta $7a
lda $9ffe
sta $7b
lda $9ffd
sta $14
lda $9ffc
sta $15
rts                      ; zurück nach BASIC
```

Das Programm, das nachstehend gelistet ist, besteht aus zwei Teilen:

Der erste Teil von Zeile 1 bis 70 muß nur einmal beim Programmstart angesprungen werden. Es ist wichtig, daß es auch wirklich am Beginn Ihres Programmes steht, also nicht auf andere Zeilennummern übertragen wird, da sonst in den Zeilen 1 und 6 Ihre Variablen zerstört werden.

Der zweite Teil schließlich produziert aus der in CA$ eingegebenen Rechenvorschrift eine BASIC-Zeile und führt diese aus. Das Ergebnis steht in RE.

Falls sich die Operation nicht laufend ändert, können Sie natürlich auch die Zeile 50100 mehrmals hintereinander anspringen.

Ihr Programm, in das diese Routinen eingebunden sind, darf nur bis Zeile 49999 gehen. Die Zeilen ab 50000 müssen unbedingt die letzten im Programm sein.

Zur Anwendung sei nur gesagt, daß Sie auf die Eingabeanforderung (INPUT CA$) nur Funktionen eingeben brauchen, z.B. '75/2*V1-V2+SQR(V3)'.

Die Zuordnung des Resultats zu RE ist bereits in Zeile 50050 gegeben.

ACHTUNG! Wenn das Programm einmal bereits gstartet wurde, darf es nicht mehr verändert werden. Sie sollten in dem Falle NEW eingeben, das Programm neu laden und dann ändern.

Das rührt daher, daß die erzeugten Zeilen nicht unmittelbar hinter dem übrigen Programm stehen, sondern ganz oben im Speicher. Eine derartige Anordnung kann das Betriebssystem bei Einfügen oder Löschen einer Zeile nicht verwalten.

Hier nun das Programmlisting:

```
1 poke56,158:clr
2 ifpeek(45)+2>255thenpoke45,2-(256-peek(45)): poke46,peek(46)+1:goto6
4 poke45,peek(45)+26 poke47,peek(45):poke48,peek(46):poke49,peek(45):
  poke50,peek(46)
8 tm=40448
10 poketm,0:poketm+1,7:poketm+2,158:poketm+3,180 poketm+4,195
12 poketm+5,153:poketm+6,0:poketm+7,13:poketm+8,158: poketm+9,190
14 poketm+10,195:poketm+11,142:poketm+12,0: poketm+13,0:poketm+14,0
20 vl=peek(45):vh=peek(46):vt=vh*256+vl
30 pokevt-4,1:pokevt-3,158:pokevt-2,179:pokevt-1,195
32 data165,122,141,255,159,165,123,141,254,159,165,20,141,253,159,165,21
33 data141,252,159,169,11,133,122,32,121,165
34 data162,0,189,0,2,240,6,157,5,158,232,208,245,169,58,157,5,158
36 data169,142,157,6,158,169,0,157,7,158,157,8,158,157,9,158
40 data173,255,159,133,122,173,254,159
50 data133,123,173,253,159,133,20,173,252,159,133,21,96
60 fori=40704to40785
70 read mc:pokei,mc:nexti
```

```
50000 rem      calculator  ****************************
50040 bu=523:input"calc";ca$
50050 pokebu,asc("r"):pokebu+1,asc("e"):pokebu+2,asc("="):bu=bu+2
50060 fori=1tolen(ca$)
50070 pokebu+i,asc(mid$(ca$,i,1)):nexti
50073 bc=len(ca$)+1
50075 pokebu+bc,0:pokebu+bc+1,0:pokebu+bc+2,0
50080 sys40704
50095 gosub50100
50097 return
```

Zum (hoffentlich) guten Schluß noch eine Anregung zum Nachdenken:

Stellen Sie sich vor, Sie haben ein Verfahren gefunden, das aus einer speziell zu formulierenden Problematik die zur Lösung erforderliche BASIC-Zeile erzeugt. Stellen Sie sich weiter vor, dieses Verfahren sei so universell, daß es in der Lage ist, aus einer Kette von Aufgaben auch eine Reihe von BASIC-Zeilen zu generieren einschließlich aller Schleifen und Sprünge.

Bleibt nur noch das Problem, diese Zeilen hintereinander in den Speicher zu bringen. Das oben beschriebene Verfahren kann zwar nur eine BASIC-Zeile erzeugen, jedoch ließe es sich so erweitern, daß es auch für mehrere Zeilen taugt.

Zu beachten ist hierbei, daß das Maschinenprogramm die jeweils erzeugte Zeile nicht immer an dieselbe Stelle transferiert, sondern die Startadresse entsprechend der Länge der zuvor erzeugten Zeile heraufsetzen muß.

Außerdem müßten die Zeilenfolgepointer (das sind die ersten beiden Bytes am Beginn jeder BASIC-Zeile) entsprechend versorgt werden, was wir in unserem Beispiel vernachlässigt haben, da der Rücksprungbefehl in der erzeugten Zeile selbst steht.

Vielleicht schaffen Sie es, mit dieser Anregung einen wirklich guten Programmgenerator zu bauen, denn um einen solchen handelt es sich im Prinzip. Programmgeneratoren sind nämlich

Programme, die mit bestimmten Aufgaben aus einem speziellen Themenkreis betraut werden und daraus ein Programm in einer bekannten Programmiersprache (es muß nicht immer BASIC sein) erzeugen, also eine Art Macrointerpreter.

1.22 Datazeilengenerator

Noch zwei Beispiele eines solchen Macrointerpreters; dabei vereinfachen wir das Prinzip der Erzeugung einer BASIC-Zeile, indem wir den Tastaturbuffer zur Hilfe nehme. Das hat den Nachteil, daß dann sämtliche Variablen gelöscht werden.

Zunächst biete ich Ihnen ein Programm an, mit dem Sie die Assembler-Datas für einen BASIC-Lader generieren können. Sie brauchen dabei nichts weiter zu tun, als die Start- und Endadresse Ihres Programms sowie die Anfangszeilnummer und die Schrittweite einzugeben. Hier nun das Programm, das, wie schon gesagt, vollkommen in BASIC geschrieben ist:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***   datazeilengenerator      ***
130 rem ***  ----------------------    ***
140 rem ***                            ***
150 rem ********************************
160 :
170 print chr$(147);
180 input "startadresse   ";ad
190 input "endadresse     ";ed
200 input "startzeilnummer";zn
220 input "schrittweite   ";sw
230 :
240 print chr$(147);zn;"data ";
250 :
260 for x=ad to ad+6
270 :a$=str$(peek(x))
280 :a$=right$(a$,len(a$)-1)
290 :a$=left$("   ",3-len(a$))+a$
```

```
300 :print a$+",";
310 next x
320 print chr$(20)
330 print "zn=";zn;":sw=";sw;
340 print ":ad=";ad;chr$(157)+":ed=";ed;
350 print ":goto410"
360 poke631,19
370 for x=632 to 635
380 :poke x,13
390 next
400 poke198,5:end
410 ad=ad+7:zn=zn+sw
420 if ad<ed then 240
430 :
440 zn=100
450 :
460 print chr$(147);:for x=zn to zn+50 step 10
470 :print x
480 next x
490 print "zn=";x;":if zn<520 then goto460"
500 poke631,19:forx=632to640:pokex,13:next:poke198,10:end
```

1.23 Bildschirmmaskengenerator

Sicher waren Sie auch schon oft in der Situation, eine bestimmte Bildschirmmaske erstellen zu müssen. Jeder weiß, was dies für eine Arbeit ist. Nun bieten wir Ihnen im Folgenden einen Bildschirmmaskengenerator, mit dem Sie nur noch Ihr Bild erstellen brauchen, und der Ihnen auf Knopfdruck Ihr erstelltes Bild in PRINT-Zeilen umwandelt. Auch dieses Programm ist wieder vollkommen in BASIC geschrieben:

```
10 rem ************************************
20 rem ***                                ***
30 rem ***  copyright (c) 1986            ***
40 rem ***                                ***
50 rem ***          by                    ***
60 rem ***                                ***
70 rem ***  klaus loeffelmann             ***
80 rem ***  lippstadt 16                  ***
90 rem ************************************
100 dimf$(15):fori=0to15:readf$(i):next
110 dima$(24),l(24):x=0:y=0:rf=0:cf=0:v=10
24:cg=55296:cl=5
120 poke53280,0:poke53281,0:poke650,128
130 goto1430
140 :
150 rem *** seite erstellen ***
160 :
170 print"■wieviel zeilen sollen bearbeite
t werden"
180 input az
190 ifaz>24thenprint"zu gross":goto 180
200 az=az-1
210 print"■■■";
220 :
230 rem tastertur abfrage
240 :
250 geta$
260 ifa$="■"thenx=x+1:printa$;:goto250
270 ifa$="■"thenx=x-1:printa$;:goto250
280 ifa$=chr$(13)andy<23thenprinta$;:x=0:y
=y+1:goto250
290 ifa$=chr$(13)andy=24then250
300 ifa$="■"andy>0thenprinta$;:y=y-1:goto2
50
310 ifa$="■"andy<=0then250
320 ifa$="■"andy<24thenprinta$;:y=y+1:goto
250
330 ifa$="■"andy>=0then250
340 ifa$="■"thenprinta$;:x=0:y=0:goto250
350 ifa$=chr$(20)andx<22andx>0thenprinta$;
:x=x-1:goto250
```

```
360 ifa$=chr$(20)then250
370 ifa$="■"thenprinta$;:x=0:y=0:goto250
380 ifa$="■"then250
390 ifa$="■"thenprinta$;:c1=0:goto250
400 ifa$="■"thenprinta$;:c1=1:goto250
410 ifa$="■"thenprinta$;:c1=2:goto250
420 ifa$="■"thenprinta$;:c1=3:goto250
430 ifa$="■"thenprinta$;:c1=4:goto250
440 ifa$="■"thenprinta$;:c1=5:goto250
450 ifa$="■"thenprinta$;:c1=6:goto250
460 ifa$="■"thenprinta$;:c1=7:goto250
470 ifa$="■"thenprinta$;:c1=8:goto250
480 ifa$="■"thenprinta$;:c1=9:goto250
490 ifa$="■"thenprinta$;:c1=10:goto250
500 ifa$="■"thenprinta$;:c1=11:goto250
510 ifa$="■"thenprinta$;:c1=12:goto250
520 ifa$="■"thenprinta$;:c1=13:goto250
530 ifa$="■"thenprinta$;:c1=14:goto250
540 ifa$="■"thenprinta$;:c1=15:goto250
550 ifa$="■"thenprinta$;:goto250
560 ifa$="■"thenprinta$;:goto250
570 ifa$="■"thenc1=c1+1:c1=abs(c1)andnot24
0:poke53280,c1
580 ifa$="■"thenc1=c1-1:c1=abs(c1)andnot24
0:poke53280,c1
590 ifa$="■"thenc2=c2+1:c2=abs(c2)andnot24
0:poke53281,c2
600 ifa$="■"thenc2=c2-1:c2=abs(c2)andnot24
0:poke53281,c2
610 ifa$="■"then810
620 :
630 printa$;:ifa$<>""thenx=x+1
640 ifx>39thenx=0:y=y+1
650 ifx<0thenx=39:y=y-1
660 ify>24theny=0
670 ify<0theny=24
680 :
690 ps=v+x+40*y:s=peek(ps):cs=cg+x+40*y:zc
=peek(cs)
```

```
700 ifs>=128thenpokeps,peek(ps)andnot128:f
=1
710 ifs<128thenpokeps,peek(ps)or128:f=0
720 pokecs,cl
730 forp=1to50:geta$:ifa$=""thennext
740 iff=0thenpokeps,peek(ps)andnot128
750 iff=1thenpokeps,peek(ps)or128
760 pokecs,zc:ifa$<>""then260
770 forp=1to50:geta$:ifa$=""thennext
780 ifa$=""then250
790 goto260
800 :
810 rem uebernehmen
820 for y=0 to az
830 for x=39 to 0 step-1
840 bs=peek(v+x+40*y)
850 ifbs<>32then1(y)=x:next y:goto880
860 poke v+x+40*y,160:pokecg+x+40*y,5
870 next x:1(y)=0:nexty
880 :
890 rem in ascii-code wandeln
900 cf=0:rf=0
910 fory=0toaz
920 a$(y)=a$(y)+"print"
930 if1(y)=0thennexty:goto1100
940 a$(y)=a$(y)+chr$(34)
950 for x=0to1(y)
960 bc=peek(v+x+40*y):ifbc=34thenbc=39
970 fc=peek(cg+x+40*y)andnot240:gosub 1350
980 :
990 rem auf revers testen+uebrnehmen
1000 ifbc>128andrf=0thenrf=1:a$(y)=a$(y)+"
■":bc=bc-128:goto1030
1010 ifbc>128andrf=1thenbc=bc-128:goto1030
1020 ifbc<128andrf=1thenrf=0:a$(y)=a$(y)+"
■"
1030 :
1040 ifbc<32thena$(y)=a$(y)+chr$(bc+64)
1050 ifbc>31andbc<64thena$(y)=a$(y)+chr$(bc)
```

```
1060 ifbc>63andbc<96thena$(y)=a$(y)+chr$(b
c+32)
1070 ifbc>96thena$(y)=a$(y)+chr$(bc+64)
1080 nextx:fori=0toaz:ifl(i)=40thena$(i)=a
$(i)+";"
1090 rv=0:a$(y)=a$(y)+chr$(34):next y
1100 print"einen moment bitte"
1110 z=0:forz1=0toaz
1120 forz2=1tolen(a$(z1))
1130 poke49152+z,asc(mid$(a$(z1),z2,1))
1140 z=z+1:next z2
1150 poke49152+z,0:z=z+1
1160 nextz1:goto1440
1170 :
1180 rem *** uebernehmen ***
1190 :
1200 print"
1210 input "zeilnummer (>5000)";zn
1220 ifzn<5000then1200
1230 input "schrittweite";sw:z=49152
1240 print";:ifza>azthen1330
1250 printzn;
1260 a=peek(z):z=z+1
1270 ifa=0thenprint:goto1300
1280 printchr$(a);
1290 goto1260
1300 za=za+1
1310 print"zn=";zn+sw;":sw=";sw;":z=";z;
":za=";za;":az=";az;":goto1240
1320 poke631,19:fori=632to640:pokei,13:nex
t:poke198,10:end
1330 run
1340 :
1350 rem farben setzen
1360 if bc=32 thenreturn
1370 iffc<>caandbc<>32thena$(y)=a$(y)+f$(f
c)
1380 ca=fc:return
1390 :
```

```
1400 rem farbdatas
1410 :
1420 data"■","■","■","■","■","■","■","■","■","■","■","■","■","■","■","■"
1430 rem menue
1440 print"■"
1450 print"menue :■■■"
1460 print"■■seite erstellen.............1
"
1470 print"■■seite uebernehmen...........2
"
1480 print"■■seite abspeichern...........3
"
1490 print"■■seite laden.................4
"
1500 print"■■programm loeschen...........5
"
1510 :
1520 geta$:a=val(a$):ifa<1ora>5then1520
1530 onagoto150,1180,1550,1700,1840
1540 :
1550 rem *** seite abspeichern ***
1560 :
1570 print"■■bitte diskette einlegen"
1580 print"■■■■return■"
1590 wait198,255:geta$
1600 ifa$<>chr$(13)then1590
1610 input"name der seite";n$
1620 open1,8,2,n$+",p,w"
1630 print#1,az
1640 forz=0toaz
1650 print#1,a$(z)
1660 nextz
1670 close1
1680 goto1430
1690 :
1700 rem *** seite laden ***
1710 :
1720 print"■■bitte diskette einlegen"
```

```
1730 print"■■■■return"
1740 wait198,255:geta$
1750 ifa$<>chr$(13)then1740
1760 input"name der diskette";n$
1770 open1,8,2,n$+",p,r"
1780 input#1,az
1790 forz=0toaz:a$(z)=""
1800 get#1,z$:ifz$=chr$(13)thennext:close1
:goto1430
1810 a$(z)=a$(z)+z$:goto1800
1820 :
1830 rem *** programm loeschen ***
1840 :
1850 zn=10
1860 print"■";
1870 forz=zntozn+70step10
1880 printz
1890 nextz
1900 :
1910 print"zn=";z":ifzn<1910 then 1860"
1920 poke631,19:fori=632to640:pokei,13:nex
t:poke198,10:end
```

Und nun die Beschreibung des Bildschirmmaskengenerators:

Nach dem Starten des Programms mit "RUN" meldet sich der Maskengenerator zunächst mit folgendem Menü:

1.....Seite erstellen

2.....Seite übernehmen

3.....Seite laden

4.....Seite abspeichern

5.....Programm löschen

Sie können nun durch Drücken der Zahlentasten die einzelnen Menüpunkte anwählen.

1.) Seite erstellen

In diesem Menüpunkt können Sie Ihre Seite ganz nach Ihren Vorstellungen aufbauen. Dabei stellt Ihnen das Programm die ganz normalen Editorfunktionen zur Verfügung. Nachdem Sie Ihr Kunstwerk erstellt haben, können Sie den Editor mit "CONTROL/B" wieder verlassen.

2.) Seite übernehmen

Durch diese Funktion wird Ihre Kreation in PRINT-Zeilen übernommen. Sie brauchen nur noch die Nummer der Startzeile und die Schrittweite der Zeilennummern anzugeben. Den Rest erledigt das Programm.

3.) Seite abspeichern

Die erstellte Seite wird durch Anwahl dieses Punktes auf Diskette abgespeichert. Zuvor müssen Sie dieser Datei noch einen Namen geben.

packen wir gleich noch das Kopieren des BASIC-ROMs ins RAM mit hinein. Wir haben uns dadurch die langsame BASIC-POKE-Schleife erspart.

Wir haben die kleine Routine in den Kassettenpuffer gelegt. Nach dem Eingeben oder Laden wird sie einmal mit SYS 828 aufgerufen und ab sofort läßt sich der LIST-Befehl ohne Programmabbruch verwenden.

```
105:  033c                    *=   828     ; Kassettenpuffer
110:  033c             chrptr =   $7a      ; Programmzeiger
112:  033c             chrgot =   $79      ; holt letztes Zeichen
114:  033c             list   =   $a69c    ; LIST-Routine
115:  033c             lstvec =   $a042    ; Zeiger auf LIST-Routine
116:  033c             nextst =   $a8f8    ; setzt Programmzeiger
117:  033c             crlf   =   $aad7    ; gibt CR aus
119:  033c a2 20               ldx  #32     ; 32 Pages kopieren
120:  033e a9 a0               lda  #>$a000 ; Zeiger auf BASIC-Anfang
120:  0340 a0 00               ldy  #0
120:  0342 84 22               sty  $22
120:  0344 85 23               sta  $23
121:  0346 b1 22       loop    lda  ($22),y ; Transferschleife
121:  0348 91 22               sta  ($22),y
121:  034a c8                  iny
122:  034b d0 f9               bne  loop
122:  034d e6 23               inc  $23
122:  034f ca                  dex
122:  0350 d0 f4               bne  loop
123:  0352 a9 60               lda  #$60    ; RTS-Kode
123:  0354 8d 14 a7            sta  $a714
123:  0357 a9 ea               lda  #$ea    ; NOP-Kode
123:  0359 8d bb a6            sta  $a6bb
123:  035c 8d bc a6            sta  $a6bc
125:  035f a9 6d               lda  #<newlst-1 ; Zeiger auf neue LIST-
                                               ; Funktion
125:  0361 8d 42 a0            sta  lstvec
126:  0364 a9 03               lda  #>newlst-1
126:  0366 8d 43 a0            sta  lstvec+1
127:  0369 a9 36               lda  #$36    ; Speicher auf RAM schalten
```

```
127:   036b 85 01            sta  1
127:   036d 60               rts
129:   036e a5 7a    newlst  lda  chrptr
130:   0370 48               pha          ; Programmzeiger retten
140:   0371 a5 7b            lda  chrptr+1
150:   0373 48               pha
160:   0374 20 79 00         jsr  chrgot  ; letztes Zeichen holen
170:   0377 20 9c a6         jsr  list    ; LIST ausführen
175:   037a 20 d7 aa         jsr  crlf    ; 'CR' ausgeben
180:   037d 68               pla
190:   037e 85 7b            sta  chrptr+1
200:   0380 68               pla          ; Programmzeiger zurückholen
210:   0381 85 7a            sta  chrptr
220:   0383 20 f8 a8         jsr  nextst  ; Zeiger auf nächstes
                                          ; Statement
230:   0386 4c 79 00         jmp  chrgot  ; letztes Zeichen holen
```

Hier wieder das Ladeprogramm in BASIC.

```
100 for i = 828 to 904
110 read x : poke i,x : s=s+x : next
120 data 162, 32,169,160,160,  0,132, 34,133, 35,177, 34
130 data 145, 34,200,208,249,230, 35,202,208,244,169, 96
140 data 141, 20,167,169,234,141,187,166,141,188,166,169
150 data 109,141, 66,160,169,  3,141, 67,160,169, 54,133
160 data   1, 96,165,122, 72,165,123, 72, 32,121,  0, 32
170 data 156,166, 32,215,170,104,133,123,104,133,122, 32
180 data 248,168, 76,121,  0
190 if s <> 9613 then print "fehler in datas !!" : end
200 print "ok !"
```

Wenn Sie folgendes kleine BASIC-Programm einmal ohne und dann nach Aufruf von SYS 828 laufen lassen, sehen Sie den Unterschied.

```
100 PRINT "LIST-TEST"
110 LIST 120
120 GOTO 100
```

1.25 Berechnete Zeilennummern bei GOTO, GOSUB und RESTORE

Eine sehr nützliche Änderung des BASIC-Interpreters wollen wir uns jetzt ansehen.

Wollen Sie eine Programmverzweigung oder ein Unterprogramm aufrufen, so müssen Sie die Zeilennummer als Zahl fest eingeben. Für eine komfortablere Programmierung wäre es jedoch wünschenswert, wenn die Zeilennummer berechnet werden könnte, z.B. wie es folgendes kleine Programm annimmt.

```
100 INPUT "ZEILENNUMMER ";Z
110 GOTO Z
...
```

Dies könnte eine umfangreiche ON .. GOTO Reihe ersetzen. Gleiches gilt für den GOSUB-Befehl. Auch hier wollen wir berechnete Zeilennummern zulassen.

Ein bedeutender Fortschritt wäre auch ein RESTORE Zeilennummer. Hat man z.B. mehrere verschiedene zusammenhängende Datenblöcke, die man mehrfach lesen möchte, so konnte man bisher den READ-DATA-Zeiger immer nur zum Programmanfang zurücksetzen und mußte dann eventuell große Mengen an Daten überlesen, bis man zu den gewünschten Daten kam. Ein RESTORE Zeilennummer erlaubt Ihnen nun, den DATA-Zeiger gezielt auf jede beliebige Zeile zu setzen.

Die Änderung des GOTO-Befehls läßt sich mit ein paar POKEs bewerkstelligen. Wir brauchen lediglich den Aufruf zum Holen der Zeilennummer durch eine Routine ersetzen, die einen belie-

bigen numerischen Ausdruck holt und diesen in das erforderliche Adressformat umwandelt. Den GOSUB-Befehl haben wir damit gleich mit modifiziert, da von GOSUB die GOTO-Routine aufgerufen wird.

```
A8A0  20 C0 02   JSR $02C0

02C0  20 8A AD   JSR $AD8A
02C3  4C F7 B7   JMP $B7F7
```

Hier haben wir den zusätzliche Code an die Adresse $02C0 (704) gelegt, die normalerweise frei ist, solange Sie nicht Sprite 11 benutzen.

Der Code läßt sich mit ein paar Pokes in den Speicher bringen

```
100 FOR I=40960 TO 49151
110 POKE I, PEEK(I) : NEXT
120 A=10*4096+8*256+10*16
130 FOR I=A TO A+2
140 READ X : POKE I,X :NEXT
150 A=704
160 FOR I=A TO A+5
170 READ X : POKE I,X :NEXT
200 DATA 32,192,2
210 DATA 32,138,173,76,247,183
```

Damit haben wir den GOTO- und GOSUB-Befehl abgehandelt. Beim RESTORE-Befehl ist etwas mehr erforderlich, da normalerweise gar keine Zeilennummer erlaubt ist. Wir müssen also unterscheiden, ob es sich um einen RESTORE-Befehl mit oder ohne Zeilennummer handelt. Dies ist jedoch nicht schwierig.

```
02C6 D0 03      BNE $02CB   ; folgen weitere Zeichen?
02C8 4C 1D A8   JMP $A81D   ; zum alten RESTORE-Befehl
02CB 20 C0 02   JSR $02C0   ; Zeilennummer holen
02CE 20 13 A6   JSR $A613   ; Adresse der Zeilennummer berechnen
02D1 38         SEC
02D2 A5 5F      LDA $5F     ; Adresse low
02D4 E9 01      SBC #$01    ; eins abziehen
02D6 A4 60      LDY $60     ; Adresse high
02D8 4C 24 A8   JMP $A824   ; weiter wie alter RESTORE-Befehl
```

Den Code können wir wieder durch POKEs eingeben.

```
300 A=2*256+12*16+6
310 FOR I=A TO A+20
320 READ X : POKE I,X : NEXT
330 DATA 208,3,76,29,168,32,192,2,32,19,166
340 DATA 56,165,95,233,1,164,96,76,36,168
```

Jetzt müssen wir dem Interpreter noch mitteilen, wo die neue RESTORE-Routine steht:

```
400 POKE 40996, 197 : POKE 40997, 2
410 POKE 1,54
```

Zeile 410 schaltet wieder auf RAM um. Ab sofort können Sie den RESTORE-Befehl in drei Varianten benutzen:

Einmal ohne Zeilennummer wie bisher, dann mit einer Zeilennummer sowie mit einem Ausdruck, der ausgewertet wird und die Zeilennummer ergibt. In diesen Fall wird beim nächsten READ-Befehl das DATA-Element gelesen, das als nächstes ab dieser Zeilennummer steht. Existiert die Zeilennummer nicht, erfolgt keine Fehlermeldung "UNDEF'D STATEMENT", sondern der Zeiger wird automatisch auf die nächste Zeile gesetzt. Jetzt wäre z.B. folgende Programmierung möglich.

```
100 GOTO 200
...
200 RESTORE 10
...
500 RESTORE
...
800 GOSUB A*2+100
...
900 RESTORE X*100+500
```

1.26 Der Kassettenpuffer als Programmspeicher

Will man kleine Maschinenprogramme zusammen mit BASIC verwenden, stellt sich immer die Frage in welchem Speicherbereich man solche Programme ablegt.

Bei der Auswahl muß beachtet werden, daß dieser Bereich nicht von BASIC-Programmen oder Variablen überschrieben werden darf. Unter diesem Gesichtspunkt gibt es zwei grundsätzliche Möglichkeiten zum Ablegen von Maschinenprogrammen.

Zum ersten kann man Speicherbereiche auswählen, die vom BASIC-Interpreter nicht benutzt werden, zum zweiten ist das Verlegen des BASIC-Programmstarts oder des Endes des BASIC-Speichers möglich.

Unbenutzt von BASIC sind wieder drei Bereiche. Als erstes bietet sich der Kassettenpuffer an. Er liegt von Adresse 828 bis 1019 ($033C bis $03FB). Dieser Bereich wird vom Programm nur dann benutzt, wenn Sie Daten mit dem Kassettenrekorder abspeichern oder laden. Ansonsten ist dieser Bereich unbenutzt. Er eignet sich damit hervorragend zur Speicherung kleiner Maschinenprogramme bis 192 Bytes Länge. Falls Sie die Sprites 13, 14 oder 15 benutzen, ist der Kassettenpuffer durch die Sprites belegt. Ein weiterer kleiner Bereich steht Ihnen von Adresse 704 bis 767 ($02C0 bis $02FF) zur Verfügung (64 Bytes). Dieser Bereich wird jedoch von Sprite 11 belegt. Einen großen Bereich von 4 KBytes haben Sie im RAM oberhalb des

BASIC-Interpreters von Adresse 49152 bis 53247 ($C000 bis $CFFF), der auch für längste Maschinenprogramme ausreichen dürfte.

Braucht man lediglich ein paar Speicherzellen, um z.B. Variablenwerte abzulegen, die auch nach einem CLR- oder NEW-Befehl noch erhalten bleiben sollen, so hat man dazu 16 Bytes 'hinter' dem Bildschirmspeicher zur Verfügung. Der 64er hat für den Bildschirmspeicher 1 KByte = 1024 Byte zur Verfügung; für das Video-RAM werden jedoch nur 25*40 = 1000 Bytes benötigt. 24 Bytes sind also übrig. Davon werden jedoch die letzten 8 Bytes als Zeiger für die Sprites benutzt. Es bleiben also 16 Bytes, die Sie für Ihre Zwecke nutzen können. Es sind dies die Adressen 2024 bis 2039 (hexadezimal $07E9 bis $07F8).

Sind die oben beschriebenen Bereiche belegt oder brauchen Sie mehr Speicherplatz, bietet es sich an, den Bereich für BASIC-Programme einzuschränken und damit für Maschinenprogramme nutzbar zu machen. Hierbei können Sie das BASIC-Ende runtersetzen (die allgemein übliche Methode) oder den BASIC-Start nach oben legen. Sehen wir uns jetzt an, wie man dies macht.

Der BASIC-Interpreter hat zwei Zeiger, die jeweils auf Start und Ende des BASIC-Speichers zeigen. Der BASIC-Start steht in 43/44 ($2B/$2C), das Ende in 55/56 ($37/$38). Mit

```
PRINT PEEK(43)+256*PEEK(44)  bzw.
PRINT PEEK(55)+256*PEEK(56)
```

lassen sich die Werte holen; im Normalfall steht dort 2049 bzw. 40960. Wollen wir Platz für ein 1000 Byte langes Maschinenprogramm schaffen, können wir das BASIC-Ende um 1000 nach unten legen. Das Ende müßte also jetzt bei 39960 liegen. Jetzt können wir den neuen Wert mittels POKE setzen.

```
HB = INT (39960/256) : LB = 39960 - HB*256
POKE 55,LB : POKE 56,HB : CLR
```

Der CLR-Befehl ist unbedingt erforderlich, da Sie sonst falsche Variablenwerte erhalten können. Wollen wir den Start auf 3049 legen, sind folgende Befehle erforderlich:

```
HB = INT (3049/256) : LB = 3049 - HB*256
POKE 43, LB : POKE 44,HB : POKE 3049-1,0 : NEW
```

Hier ist der NEW-Befehl erforderlich, um alle anderen BASIC-Zeiger richtig zu setzen.

1.27 Bubblesort in Assembler

Eine Aufgabe, auf die jeder Programmierer des öfteren stößt, ist das Sortieren irgendwelcher Daten. Das können z.B. Namen, Adressen und Zahlenreihen sein. Zum Sortieren gibt es verschiedene bekannte Algorithmen, die jedoch alle sehr zeitaufwendig sind, sobald eine größere Anzahl von Daten zu sortieren sind. Die einfachsten Verfahren sind im allgemeinen auch die langsamsten. Braucht man entschieden schnellere Sortierverfahren, so muß man die Algorithmen nicht in BASIC, sondern in Maschinensprache formulieren. Bei solchen Aufgaben sind Lösungen in Maschinensprache gut 100mal so schnell wie eine BASIC-Routine. Das folgende Programm ist zur Sortierung eines Stringarrays geeignet. Um das Programm möglichst kurz zu halten, wurde von folgenden Voraussetzungen ausgegangen:

1. Das zu sortierende Feld muß als erstes mit DIM dimensioniert werden.

2. Nach dem letzten zu sortierenden Arrayelement muß ein Leerstring folgen.

Punkt 2 hat zudem den Vorteil, daß auch ein nur zum Teil gefülltes Array sortiert werden kann, ohne daß sämtliche Leerstrings nach dem Sortieren am Anfang des Arrays stehen.

Mit diesen Vereinbarungen wird das Programm so kurz, daß es im Kassettenpuffer Platz hat. Es wird einfach mit SYS 828 auf-

gerufen. Das Programm überprüft, ob das erste Array ein eindimensionales Stringarray ist. Ist dies nicht der Fall, wird das Maschinenprogramm direkt beendet.

```
100:  033c                    *=  828
105:  033c                    .opt p1
110:  033c a0 00              ldy #0
120:  033e b1 2f              lda ($2f),y  ; erster Buchstabe
130:  0340 30 0d              bmi l1
135:  0342 c8                 iny
140:  0343 b1 2f              lda ($2f),y  ; zweites Zeichen
150:  0345 10 08              bpl l1
160:  0347 a0 04              ldy #4
170:  0349 b1 2f              lda ($2f),y  ; Dimensionen
180:  034b c9 01              cmp #1
190:  034d f0 01              beq l2
200:  034f 60          l1     rts
210:  0350 18          l2     clc
220:  0351 a5 2f              lda $2f      ; Arraystart
230:  0353 69 07              adc #7       ; plus 7
240:  0355 85 6e              sta $6e
250:  0357 a5 30              lda $30
260:  0359 69 00              adc #0
270:  035b 85 6f              sta $6f
280:  035d a0 00       l3     ldy #0
290:  035f b1 6e              lda ($6e),y
300:  0361 f0 ec              beq l1       ; Länge null, fertig
310:  0363 85 22              sta $22
320:  0365 c8          l4     iny
330:  0366 b1 6e              lda ($6e),y
340:  0368 99 22 00           sta $22,y    ; Zeiger auf String
350:  036b c0 02              cpy #2
360:  036d d0 f6              bne l4
370:  036f a5 6e              lda $6e
380:  0371 85 71              sta $71
390:  0373 a5 6f              lda $6f
400:  0375 85 72              sta $72
410:  0377 18          l5     clc
420:  0378 a5 71              lda $71
```

```
430:  037a 69 03          adc  #3      ; drei addieren
440:  037c 85 71          sta  $71
450:  037e 90 02          bcc  l6
460:  0380 e6 72          inc  $72
470:  0382 a0 00    l6    ldy  #0
480:  0384 b1 71          lda  ($71),y
490:  0386 f0 3d          beq  l13
500:  0388 85 4d          sta  $4d     ; Länge
510:  038a c5 22          cmp  $22     ; mit erster Länge
                                       ; vergleichen
520:  038c 90 02          bcc  l7
530:  038e a5 22          lda  $22
540:  0390 85 55    l7    sta  $55     ; Vergleichslänge
550:  0392 c8       l8    iny
560:  0393 b1 71          lda  ($71),y
570:  0395 99 4d 00       sta  $4d,y
580:  0398 c0 02          cpy  #2
590:  039a d0 f6          bne  l8
600:  039c a0 00          ldy  #0
610:  039e b1 23    l9    lda  ($23),y  ; Stringvergleich
620:  03a0 d1 4e          cmp  ($4e),y
630:  03a2 f0 04          beq  l10     ; gleich, dann weiter
640:  03a4 b0 0b          bcs  l11     ; größer dann Tausch
650:  03a6 90 cf          bcc  l5      ; kleiner dann nächsten
                                       ; String
660:  03a8 c8       l10   iny
670:  03a9 c4 55          cpy  $55     ; alle Zeichen gleich ?
680:  03ab d0 f1          bne  l9
690:  03ad c4 22          cpy  $22     ; erster String länger
700:  03af b0 c6          bcs  l5      ; nein dann ok
710:  03b1 a0 02    l11   ldy  #2
720:  03b3 b1 6e    l12   lda  ($6e),y ; Stringzeiger vertauschen
730:  03b5 aa             tax
740:  03b6 b1 71          lda  ($71),y
750:  03b8 91 6e          sta  ($6e),y
760:  03ba 99 22 00       sta  $22,y
770:  03bd 8a             txa
780:  03be 91 71          sta  ($71),y
790:  03c0 88             dey
800:  03c1 10 f0          bpl  l12
```

```
810:   03c3 30 b2          bmi  l5
820:   03c5 18       l13   clc          ; Zeiger auf nächsten String
830:   03c6 a5 6e          lda  $6e
840:   03c8 69 03          adc  #3
850:   03ca 85 6e          sta  $6e
860:   03cc 90 8f          bcc  l3
870:   03ce e6 6f          inc  $6f
880:   03d0 d0 8b          bne  l3
```

Hier wieder das Ladeprogramm in BASIC.

```
100 for i = 828 to 977
110 read x : poke i,x : s=s+x : next
120 data 160,  0,177, 47, 48, 13,200,177, 47, 16,  8,160
130 data   4,177, 47,201,  1,240,  1, 96, 24,165, 47,105
140 data   7,133,110,165, 48,105,  0,133,111,160,  0,177
150 data 110,240,236,133, 34,200,177,110,153, 34,  0,192
160 data   2,208,246,165,110,133,113,165,111,133,114, 24
170 data 165,113,105,  3,133,113,144,  2,230,114,160,  0
180 data 177,113,240, 61,133, 77,197, 34,144,  2,165, 34
190 data 133, 85,200,177,113,153, 77,  0,192,  2,208,246
200 data 160,  0,177, 35,209, 78,240,  4,176, 11,144,207
210 data 200,196, 85,208,241,196, 34,176,198,160,  2,177
220 data 110,170,177,113,145,110,153, 34,  0,138,145,113
230 data 136, 16,240, 48,178, 24,165,110,105,  3,133,110
240 data 144,143,230,111,208,139
250 if s <> 17663 then print "fehler in datas !!" : end
260 print "ok !"
```

Die Geschwindigkeit des Maschinenprogramms können Sie mit einem kleinen Testprogramm demonstrieren.

Das Programm erzeugt eine vorgegebene Anzahl von Strings aus Zufallsbuchstaben mit einer ebenfalls einzugebenden maximalen Länge, zeigt sie am Bildschirm an, sortiert sie dann und gibt sie zusammen mit der Zeit für das Sortieren wieder aus.

```
100 input "anzahl, laenge ";a,l
110 dim a$(a):a=a-1
120 for i=0 to a
130 for j=1 to rnd(1)*l
140 a$(i) = a$(i)+chr$(rnd(1)*26+65)
150 next:next
160 for i=0 to a : print a$(i) : next
170 t=ti : sys 828 : t=ti-t
180 print "sortierzeit " t/60 "sekunden"
190 for i=0 to a : print a$(i) : next
```

Lassen Sie das Programm einmal mit verschiedenen Längen und Anzahl von Strings laufen, und beachten Sie die Sortierzeit. 100 Strings werden unter einer Sekunde sortiert. Ein entsprechendes BASIC-Programm würde dazu Minuten brauchen.

Wenn Sie das Programm für Ihre Zwecke einsetzen wollen, beachten Sie bitte, daß das letzte Element (bis zu dem sortiert werden soll) ein Leerstring sein muß und daß das Array als erstes dimensioniert werden muß.

Kapitel 2

Grafik für Fortgeschrittene

2.1 Die Textgrafik

Diese Möglichkeit der Erzeugung von Grafiken auf dem Commodore 64 ist wohl die einfachste und bequemste Art. Hier müssen keine Adressen berechnet oder auf verschiedene Register im Videocontroller geachtet werden. Man kann im Programm von der Tastatur aus verschiedene Symbole erzeugen. Um diese Symbole zu erreichen sind jeweils zwei Tasten zu drücken. Wenn Sie sich die Tastatur des Commodore 64 genauer ansehen, so werden Sie feststellen, daß fast jede Taste des Gerätes dreifach belegt ist. Das erste Symbol, meist ein Buchstabe, wird durch einfaches Drücken der Taste erlangt. Das Zweite durch gleichzeitiges Drücken der SHIFT-Taste und der Taste, auf dem sich Ihr gewünschtes Zeichen befindet. Das dritte Symbol ereicht man schließlich durch gleichzeitiges Drücken der C= Taste und der entsprechenden Taste.

Sie können nun durch den PRINT-Befehl schon recht hübsche Grafiken erzeugen, oder aber verschiedene Formulare, die sie z. B. in Adreßverwaltungen benutzen, grafisch schöner gestalten. Dazu ein Beispiel:

Schreiben Sie zunächst folgende BASIC-Programmzeile auf den Bildschirm:

```
100 PRINT "
```

Nun drücken sie die Commodore-Taste, halten diese und drücken dann auf den Buchstaben "A". Wie Sie sehen, erscheint die linke, obere Ecke eines Rahmens in Ihrer Programmzeile. Nun verfahren Sie genauso mit der Kombination SHIFT & "*", und dies 38mal. Jetzt sollte eine lange Linie auf dem Bildschirm zu sehen sein, die durch das Ende der Bildschirmzeile unterbrochen ist. Das soll uns jedoch nicht weiter stören. Schließlich

drücken Sie noch die Tasten C= & "S", und beenden die Zeile mit einem Anführungszeichen, einem nachfolgenden Semikolon und einem Druck auf die RETURN-Taste.

Sollten Sie nun Ihr Programm mit dem Befehl "RUN" starten, so können sie die obere Hälfte einer Umrahmung erkennen.

In der nächsten Zeile können Sie nun folgendes eingeben:

```
110 PRINT "
```

Danach verfahren Sie mit den Tasten SHIFT & "B", der Leertaste und nochmals SHIFT & "B" genauso wie in Zeile 100.
Wenn Sie Ihr Programm nun starten, so können Sie sehen, daß sich der Rahmen um schon eine Bildschirmzeile nach unten verlängert hat.

Die Zeile 110 können Sie nun eine Weile wiederholen (am besten 20mal). Ihr Rahmen wird dabei immer länger. Versuchen Sie nun einmal selbst Ihren Rahmen zu vollenden. Im folgenden finden Sie das fertige Programm:

```
100 PRINT "┌──────────────────────────────────────
──────────┐";
110 PRINT "│
          │";
120 PRINT "│
          │";
130 PRINT "│
          │";
140 PRINT "│
          │";
150 PRINT "│
          │";
160 PRINT "│
          │";
170 PRINT "│
          │";
```

```
180 PRINT "|
          |";
190 PRINT "|
          |";
200 PRINT "|
          |";
210 PRINT "|
          |";
220 PRINT "|
          |";
230 PRINT "|
          |";
240 PRINT "|
          |";
250 PRINT "|
          |";
260 PRINT "|
          |";
270 PRINT "|
          |";
280 PRINT "|
          |";
290 PRINT "|
          |";
300 PRINT "|
          |";
310 PRINT "|
          |";
320 PRINT " └──────────────────────────────────────
──────────┘";
```

Alle kommerziellen Programme werden heute durch die Anwendung solcher Rahmen sehr übersichtlich und benutzerfreundlich gestaltet. Wie Sie sehen kann auch der Commodore 64 wie die "Großen" programmiert werden.

Natürlich lassen sich diese Programme auch übersichtlicher gestalten. Dabei wird die CHR$-Funktion zu Hilfe genommen, die es gestattet, einfach über Eingabe des sogenannten ASCII (American Standard Code for Information Interchange), den Sie im Anhang Ihres Commodore-Handbuches nachschlagen können, das entsprechende Zeichen auszudrucken. Dasselbe Programm noch einmal mit CHR$-Codes:

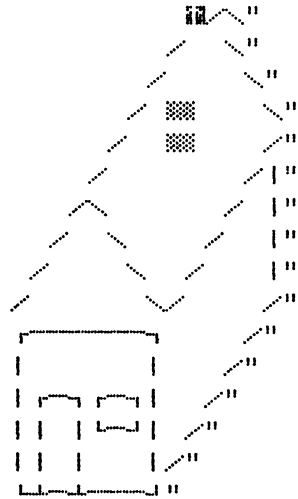
```
100 print chr$(176);
110 for z=1 to 38:print chr$(192);:next
120 print chr$(174);
130 :
140 for z=1 to 20
150 :print chr$(194);
160 :for z2=1 to 38
170 ::print chr$(32);
180 :next
190 :print chr$(194);
200 next z
210 :
220 print chr$(173);
230 for z=1 to 38:print chr$(192);:next
240 print chr$(189);
```

Zugegeben, dieses Programm ist etwas langsamer als das vorhergehende, aber dafür um so übersichtlicher und kürzer.

Doch nicht nur für gewerbliche Zwecke sind diese Grafikzeichen nützlich. Schauen Sie sich doch einmal das folgende Programm an:

```
7000 PRINT
7010 PRINT
7020 PRINT"
7030 PRINT"
7040 PRINT"
7050 PRINT"
7060 PRINT"
```

```
7070 PRINT"
7080 PRINT"
7090 PRINT"
7100 PRINT"
7110 PRINT"
7120 PRINT"
7130 PRINT"
7140 PRINT"
7150 PRINT"
7160 PRINT"
7170 PRINT"
7180 PRINT
7190 PRINT
7200 PRINT
7210 PRINT
7220 PRINT
7230 PRINT
```



Doch nicht nur so können bestimmte Zeichen auf den Bildschirm gebracht werden. Wie Sie vielleicht wissen, stellt uns der Videocontroller einen bestimmten Bildschirmbereich zur Verfügung, in dem unsere Zeichen gespeichert werden. Diese Bildschirmkodes entsprechen jedoch nicht dem uns schon bekannten ASCII. Sie können diesen Code ebenfalls in Ihrem Handbuch im Anhang nachschlagen.

Zusätzlich können Sie jedem einzelnen Zeichen eine bestimmte Farbe zuordnen. Auch diese Informationen werden in einem bestimmten Speicherbereich abgelegt. Hier nun eine Tabelle mit den verschiedenen Bildschirmspeichern:

Lage	Art des Speichers
1024 - 2023	Bildschirm-Kode Speicher
55296 - 56295	Farbspeicher der einzelnen Zeichen.

Dabei können die einzelnen Zeichen folgende Farben besitzen:

Farbe	Farbkode
schwarz	0
weiß	1
rot	2
Cyen	3
purpur	4
grün	5
blau	6
gelb	7
orange	8
braun	9
hell-rot	10
dunkel-grau	11
mittel-grau	12
hell-grün	13
hell-blau	14
hell-grau	15

Auch hier soll uns wieder unser Rahmen-Programm als Beispiel dienen:

```
100 rem ************************
110 rem *** programm: rahmen **
120 rem ************************
130 :
140 y= 0:x= 0:ch=112:f=14:gosub 340
150 y= 0:x=39:ch=110:gosub 340
160 y=24:x= 0:ch=109:gosub 340
170 y=24:x=39:ch=125:gosub 340
180 :
190 ch=64
200 for x=1 to 38
210 :
220 :    y= 0:gosub 340
```

```
230 :    y=24:gosub 340
240 :
250 next x
260 :
270 ch=93
280 for y=1 to 23
290 :
300 :    x= 0:gosub 340
310 :    x=39:gosub 340
320 next y
325 end
330 :
340 rem ***********************************
350 rem *** unterroutine zeichen setzen ***
360 rem ***********************************
370 :
380 poke  1024+x+40*y,ch
390 poke 55296+x+40*y,f
400 return
```

Wenn Sie wollen, können Sie diese kleine Unterroutine auch in Ihre eigenen Programme einbauen. Sie wird Ihnen dort sicher gute Dienste leisten.

2.2 Grafikanimation

2.2.1 Zweifarbiger Bildschirmrand

Normalerweise besitzt der C-64 lediglich einen einfarbigen Bildschirm. Doch mit minimalem Aufwand lassen sich sogar von BASIC aus mehrfarbige BORDER-Farben realisieren.

Geben Sie dazu die folgende Routine ein:

```
50 POKE 53280,1
60 FOR A=0 TO . :   NEXT
70 POKE 53280,0
80 :GOTO 50
```

Hier wird beim Umschalten der beiden Farben 0 (schwarz) und 1 (weiß) eine genaue Farbwechselperiode erzeugt, welche eine exakte Grenze zwischen den beiden verschiedenen Farben ermöglicht.

Dabei ist es wichtig, daß dieses wie auch die folgenden Programme und Routinen wirklich genau wie vorgegeben von Ihnen abgetippt werden, um die jeweils gewünschte, bzw. erforderliche Zeitperiode zu erreichen.

Obwohl das Prinzip äußerst simpel ist, lassen sich so bereits ungemein eindrucksvolle Animationen verwirklichen. Die obige Routine dient dabei mehr dem Verständnis, als einem praktischen Einsatz.

Eine derartige BORDER-Routine läßt sich am sinnvollsten einsetzen, wenn ein Programm auf einen Tastendruck warten soll. Während der Benutzer sich noch ein Menu oder eine Anleitung durchliest, läuft parallel ein BORDER-Effekt ab.

2.2.2 Multicolor-Border-Farben

Ein Beispiel einer derartigen, auf einen Tastendruck wartenden Routine ist die folgende:

```
100 A=53280
110 FORF=0TO.:::NEXT:POKE A,0:POKEA,2:POKEA,6:POKEA,7:GET
A$:IF A$=""THEN 110
```

Der hier erzielte Effekt ist trotz minimalem Aufwand beachtlich: Ihnen wird hier ein fast stehender, jedoch langsam scrollender vierfarbiger Bildschirmrand präsentiert. Sobald Sie eine Taste betätigen, wird die Routine beendet.

Das Prinzip entspricht genau dem des zweifarbigen Bildschirmrandes. Es ist lediglich etwas aufwendiger, die exakte Farbwechselperiode für vier verschiedene Bildschirmfarben herauszufinden.

Mit der nächsten Routine wird der Scroll-Effekt noch verstärkt. Geben Sie dazu die beiden folgenden Zeilen ein:

```
200 A=53280
210 FORF=0TO.:::NEXT:POKE A,0:POKEA,2:POKE A,6:POKE
A,7:GET A$:IF A$=""THEN 210
```

Über eine gelbe Grund-Borderfarbe "gleitet" hier ein schwarz-rot-blauer Multi-Farbbalken.

Dabei können Sie die Dicke der einzelnen Farbbalken durch eine geschickte Anordnung der unterschiedlichen Farbpokes noch gezielt verändern.

Wollen Sie beispielsweise den schwarzen Balken der obigen Routine vergrößern, so geschieht dies folgendermaßen:

```
300 A=53280
310 POKEA,0:FORF=0TO.:::NEXT:POKEA,2:POKE A,6:POKE A,7:GET
A$:IF A$="" THEN 310
```

Für einige Anwendungen, wie beispielsweise die Untermalung eines neuen High-Scores, ist es oft ansprechender, ein "unruhiges" Rahmenmuster zu generieren.

Die folgenden drei Zeilen bewirken, daß drei schwarze Balken gleichzeitig von oben nach unten über den Rahmen wandern, wobei parallel jeweils eine Vielzahl jeweils 1/3 so großer bunter Balken die Zwischenräume auffüllt:

```
400 A=53280
410 POKEA,0:FORF=0TO.::::NEXT:POKEA,2:POKE A,6:POKE A,7
420 POKE A,16:POKEA,5:POKE A,9:POKEA,14:GETA$:IF A$="" THEN
410
```

Und nun kommt ein weiterer Einzeiler, der einen sechsfarbigen Rahmen erzeugt:

```
A=53280

500 POKEA,1:POKEA,2:POKEA,3:POKEA,4:POKEA,7:POKEA,12::
GOTO500
```

Weitere Änderungen nach dem umseitigen Muster werden in den drei weiteren Routinen erzeugt:

A)

```
600 :   : : GET X$:IFX$=""THENPOKEA,1:POKEA,2:
POKEA,3:POKEA,4:POKEA,7:GOTO600
```

B)

```
700 POKEA,4: :GET X$:IF X$=""THENPOKEA,1:POKEA,2:
POKEA,3:::POKEA,7:GOTO700
```

C)

```
1000 FORF=0TO0STEP0:POKEA,4:GETX$:IFX$=""THENPOKEA,3:
FORK=1TO4:NEXT:POKEA,7:NEXT
```

2.2.3 Kombinierte Border- und Paperfarbe

Die bereits vorgestellten BORDER-Animationeffekte lassen sich auch auf den gesamten Bildschirm ausweiten. Probieren Sie dazu einmal den folgenden Einzeiler aus:

```
A=53280
B=53281

4000 POKE A,1:POKE B,1:POKEA,4:POKEB,4:POKEA,8:
POKEB,8:GOTO4000
```

Der mit nur geringem Aufwand erzielte Effekt ist äußerst ansprechend: Von unten nach oben scrollt hier ein heller Balken über den Bildschirm. Dabei bleiben alle Bildschirminhalte bestehen. Auf dem Rahmen ist der Farbbalken schneller beendet als im eigentlichen Bildschirmbereich. Aber probieren Sie den Einzeiler doch einfach einmal aus.

Nach derselben Methode stelle ich Ihnen nun drei ähnliche kurze Routinen vor:

A)

```
5000 POKE A,1:POKE B,1:POKEA,4:POKEB,4:POKEA,8:POKEB,8
5010 POKE A,12:POKEB,12:GOTO5000
```

B)

```
6000 POKE A,0:POKE B,0:POKEA,1:POKEB,1:POKEA,0:POKEB,0
6010 POKE A,12:POKEB,2:POKEA,15:POKEB,15:POKE
A,1:POKEB,7:POKEA,2
6020 GETA$:IFA$=""THEN6000
```

C)

```
7000 POKE A,1:POKE B,5:POKEA,4:POKEB,8:POKEA,11:POKEB,3
7010 POKE B,6
7020 FORG=1TO4:NEXT:GOTO 7000
```

Alle Ihnen hier gezeigten BORDER- und PAPER-Animationsroutinen sind - obwohl sehr simpel - doch äußerst eindrucksvoll und effektiv.

Entwickeln Sie nun ruhig nach Belieben anhand der vorgestellten Beispiele Ihre eigenen Routinen nach dem Grundmuster und bringen Sie so kräftig Farbe und Aktion in Ihre Programme!

2.3 Extended-Colour-TEC-Modus

Der Commodore besitzt bekanntlich herausragende Farbmöglichkeiten. Will man jedoch im normalen Text-Modus Farbe einsetzen, so stößt man schnell an eine scheinbar unüberwindbare Grenze. Sie können im Text-Modus lediglich die Farbe der Vordergrund-Zeichen wählen. Die Hintergrundfarbe bleibt jedoch immer gleich und kann nicht geändert werden.

Geben Sie zur Veranschaulichung einmal das folgende Programm ein:

```
10 FOR P=1 TO 2: RESTORE
15 FOR I=1 TO 8: READ A
20 PRINT CHR$(A);"ALLE 8 GRUND-FARBEN...";
30 NEXT I
40 PRINT CHR$(18);:NEXT P
100 DATA 5,28,30,31,156,158,159,144
```

Hier wird ganz deutlich, daß die Farbe des Hintergrunds immer konstant bleibt. Sie können diese zwar über die bekannte Adresse 53281 ändern, doch setzen Sie damit die gesamte Hintergrund-Bildschirmfarbe.

Der Commodore besitzt jedoch einen TEXT-EXTENDED COLOUR-MODUS (im folgenden kurz "TEC"). Im TEC haben Sie die Möglichkeit, jedem Zeichen auf dem Display eine von vier verschiedenen Hintergrundfarben zuzuordnen.

Der TEC-Modus wird über

```
POKE 53265,91
```

aktiviert. Wenn Sie diesen POKE jedoch eingeben, so werden Sie Schwierigkeiten haben, den Cursor zu entdecken. Geben Sie bei einem normalen blauen Hintergrund POKE 53283,1 ein. Die Cursorfarbe wechselt dann zu weiß.

In diesem Modus haben Sie nun wie bereits angedeutet, jeweils vier Hintergrundfarben zur Auswahl. Aber diesen Farbreichtum müssen Sie sich "erkaufen". Konkret bedeutet dies, daß der verfügbare Zeichensatz von 128 Zeichen + Reverse Characters auf magere 64 absinkt. Alle Graphikzeichen sind nicht mehr ansprechbar.

Allerdings entschädigen Sie die nun plötzlich neu möglichen Farbeffekte.

Warum ist der Zeichensatz eigentlich im TEC-Modus derart begrenzt? Sie werden benötigt, um die jeweilige Hintergrundfarbe für die einzelnen restlichen 64 Zeichen zu bestimmen. Wie dies vor sich geht, schauen wir uns nun genauer an:

Schalten Sie Ihren Computer neu ein und geben manuell die folgenden POKEs ein:

```
POKE 53265,91:REM TEC MODUS AN

POKE 53283,1 :REM REVERSE ZEICHENFARBE=1
```

Drücken Sie nun Control 9 für den "Reverse On"-Modus und geben Sie einige normale Zeichen auf dem Bildschirm ein. Sie sollten jetzt lediglich einen weißen Strich sehen.

Drücken Sie nun die Cursor-Down-Taste und RETURN.

Jetzt geben Sie nur noch

```
POKE 53283,0
```

ein, und Sie sehen etwas völlig neues! Die Hintergrundfarbe des Bildschirms ist blau, die Zeichenfarbe weiß, aber die Hintergrundfarbe des von Ihnen eingegebenen Texts ist durch den letzten POKE-Befehl auf schwarz gesetzt worden.

Diese lokale Hintergrundfarbe können Sie beliebig verändern, indem Sie in die Adresse 53283 die Farbwerte von 0 bis 15 POKEn.

Damit haben wir nach der normalen über POKE 53281 festlegbaren die zweite, unabhängige Hintergrundfarbe kennengelernt.

Halten wir kurz fest: Den TEC-Modus erreicht man über POKE 53265,91. Wenn nun Zeichen im Reverse On Modus ausgegeben werden, so läßt sich deren Hintergrundfarbe über die Adresse 53283 individuell steuern.

Ein Beispielprogramm dazu kann so aussehen:

```
10 POKE 53265,91    :REM TOC AN
20 PRINT CHR$(18);  :REM REVERSE AN
30 PRINT CHR$(144); :REM ZEICHENFARBE = SCHWARZ
40 FOR I=1 TO 10
45 PRINT "HIDI-HI! ALLES KLAR?":NEXT
50 FOR F=0 TO 15    :REM ALLE 16 FARBEN
60 :  FOR A=1 TO 500:NEXT A :REM PAUSE
70 POKE 53283,F     :REM REVERSE-LOKALE HINTERGRUNDFARBE
SETZEN
80 NEXT F
```

Die dritte, von den beiden vorigen völlig unabhängige Hintergrundfarbe können Sie ansprechen, indem Sie die mit dieser Farbe zu versehenden Zeichen als SHIFTed Characters eingeben.

Die jeweilige Hintergrundfarbe wird über die Adresse 53282 bestimmt.

Schauen Sie sich zur Verdeutlichung einmal die Auswirkungen des folgenden DEMOs an:

```
10 poke 53272,23    : rem kleinschrift
15 print chr$(5);   : rem zeichenfarbe=weiß
20 for i=1 to 10:print "TEC-MODUS!": rem alle zeichen als
SHIFTed character eingeben
30 next i
40 for f=0 to 15
50 poke 53282,f  : rem shifted hintergrund setzen
60 next
```

Nun fehlt lediglich noch die vierte lokale TEC-Hintergrundfarbe. Diese wird über die Adresse 53284 jeweils individuell

festgelegt und spricht den lokalen Hintergrund aller Zeichen an, die gleichzeitig als SHIFTed und REVERSE Zeichen eigegeben wurden.

Das entsprechende DEMO sieht parallel zu den vorigen Programmen aus:

```
10 POKE 53272,23: PRINT CHR$(28);CHR$(18);
20 FOR I=1 TO 10: PRINT "TEC-FARBE NR.4!" : REM ALLE
ZEICHEN ALS SCHIFTED CHARACTER
30 NEXT I
40 FOR F=0 TO 15:POKE 53284,F
50 NEXT F
```

Bis jetzt haben wir die einzelnen TEC-Modi jeweils des besseren Verständnisses halber getrennt erörtert. Um Ihnen einen Eindruck davon zu vermitteln, wie leistungsfähig der TEC-Modus ist, geben Sie einmal das folgende DEMO ein, welches alle vier Hintergrundfarben getrennt setzt und ändert.

```
5 print chr$(147);
10 poke 53280,2
20 poke 53281,0
30 poke 53272,23
40 poke 53265,91
45 :
50 a$(1)="■ tec-modus 1 ":rem normal
60 a$(2)="■■ tec-modus 2 ■":rem reverse
70 a$(3)="■ TEC-MODUS 3 ":rem shifted
80 a$(4)="■■ TEC-MODUS 4 ■":rem revers+shi
fted
90 :
100 for i=1 to 76
110 n=rnd(1)*4+1
120 print a$(n);
130 next i
150 :
```

```
180 for f=15 to 0 step-1:gosub 1000
190 poke 53281,f:next
200 for f=0 to 13:gosub 1000
210 poke 53282,f:next
220 for f=0 to 14:gosub 1000
230 poke 53283,f:next
240 for f=0 to 15:gosub 1000
250 poke 53284,f:next
300 :
400 get a$:ifa$=""then 400
410 poke 53265,27
500 stop
600 :
1000 for t=1 to 200:next:return
```

Der TEC-Modus wird mittels

```
POKE 53265, 27
```

wieder beendet.

2.3.1 Die TEC-Referenzliste

Da es nur durch mühseliges Probieren möglich ist, Sonderzeichen wie ",$,%,#,*,... in den TEC-Modi 3 und 4 darzustellen, gebe ich Ihnen hier eine Referenztabelle:

Im TEC-Modi darzustellendes Zeichen	Einzugeben als ...
"a" bis "z"	.... SHIFT "a" bis "z"
!	..... COMMODORE K
$	..... COMMODORE KLAMMERAFFE
%	..... COMMODORE G
'	..... COMMODORE ^
(	..... COMMODORE ENGL.PFUND
#	..... COMMODORE T
=	..... COMMODORE X
/	..... COMMODORE P
PFUND	..... COMMODORE -
?	..... COMMODORE B
-	..... COMMODORE Z
+	..... COMMODORE Q
*	..... COMMODORE N
.	..... COMMODORE S
&	..... COMMODORE +
;	..... COMMODORE F
,	..... COMMODORE D
<	..... COMMODORE C
>	..... COMMODORE V
"	..... COMMODORE I
0	..... COMMODORE A
1	..... COMMODORE E
2	..... COMMODORE R
3	..... COMMODORE W
4	..... COMMODORE H
5	..... COMMODORE J
6	..... COMMODORE L
7	..... COMMODORE Y
8	..... COMMODORE U
9	..... COMMODORE O

Die Anwendungsmöglichkeiten des TEC-Modus sind äußerst vielfältig. Sehr effektiv lassen sich die hier vorgestellten Techniken in einer GEM-Simulation auf dem C-64 anwenden.

Sie bilden die einzige Möglichkeit, unterschiedliche Hintergrundfarben für die einzelnen Windows zu erzeugen.

2.4 Die hochauflösende Grafik

Unter der hochauflösenden Grafik verstehenden wir die Ansteuerung einzelner Punkte innerhalb des Bildschirms. Bei dem Commodore 64 wird dort ein extra Speicherbereich angelegt, in dem dann das Aussehen des Bildschirms gespeichert ist. Da es keine Befehle zur HRG-Grafik, so die Kurzform dieser Grafikart, gibt, müssen wir den POKE, bzw. den PEEK-Befehl zur Hilfe nehmen, die es uns gestatten, direkt in den Speicher zu schreiben oder aus diesem zu lesen.

Zunächst gilt es dem Videocontroller, das ist das Bauteil im Computer, das für den Bildschirmaufbau zuständig ist, mitzuteilen, das wir ab sofort nicht mehr im Textmodus, sondern in der HRG-Grafik arbeiten wollen. Dies geschieht über spezielle Speicherregister, die die Inhalte der Controller-register beinhalten.

In unserem Fall bräuchten wir die Register 17 und 24, in denen wir dem Videocontroller den Grafikmodus und die Lage der Grafikseite mitteilen.

Durch das folgende Programm können wir nun unsere Grafik einschalten und den Grafikspeicher löschen.

```
10 v=53248          : rem basisadresse des videocontrollers
20 POKE V+17,59     : rem hrg einschalten
30 poke V+24,24     : rem hrg nach 8192
40 :
50 for x=8192 to 16383
60 :poke x,0
70 next x
```

Dazu muß noch gesagt werden, daß die Farbgebung ab sofort von dem Speicherbereich übernommen wird, der uns zuvor als Bildschirmspeicher diente. Erweitern wir unser Programm also um die folgenden Zeilen:

```
80 :90 for x=1024 to 2023
100 :poke x,hf*16+pf
110 next x
```

2.4.1 Punktberechnung in der HRG-Grafik

Damit der VIC nun weiß, wie die Grafik aussehen soll, muß diese in dem Grafikspeicher abgelegt werden. Da die maximale Auflösung 320*200 Punkte, also insgesamt 64000 Punkte beträgt, ist es notwendig, eine spezielle Kodierung der Punktmatrix zu schaffen. Denn würde jeder Punkt ein Byte belegen, hätte der Grafikspeicher schon einen Umfang von 64K. Da bei einem 64K großen Grafikspeicher kein Platz mehr für BASIC-Programme oder den Farbspeicher bleiben würde, ist der Grafikspeicher folgendermaßen kodiert:

Jeweils acht Punkte werden von einem Byte festgelegt, was zur Folge hat, daß der Grafikspeicher nur noch ein Achtel der Größe hat, die er hätte, wenn jeder Punkt durch ein Byte dargestellt würde.

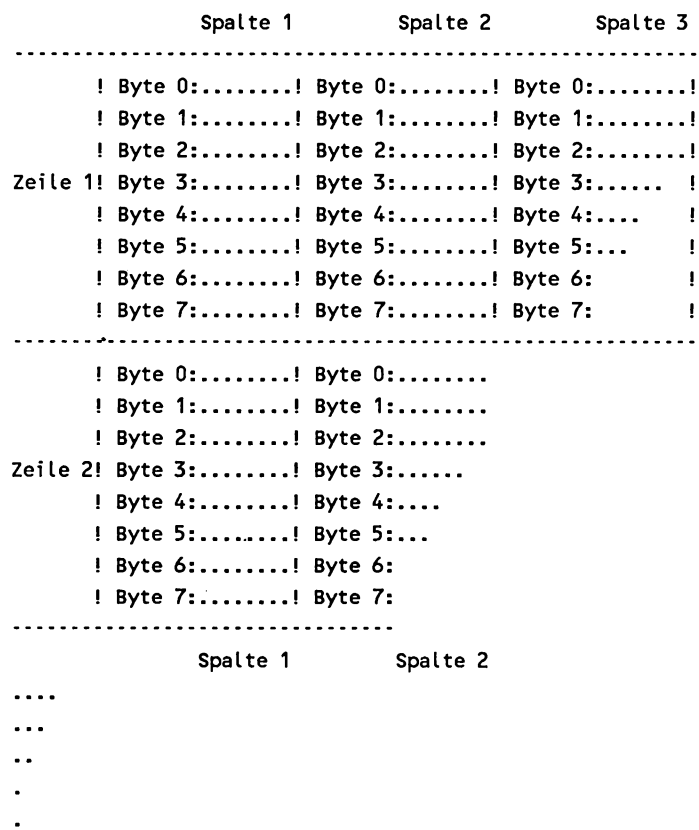
Nun wäre es durchaus logisch und praktikabel, wenn diese Bytes eines an dem anderen liegen würden, also etwa so:

```
   Byte 0          Byte 1
7 6 5 4 3 2 1 0 - 7 6 5 4 3 2 1 0 - 7 6 5 u.s.w.
```

Leider ist dies nicht so, da diese Anordnung mit der Farbgebung recht schwierig zu koordinieren gewesen wäre. Praktikabler aber war die Anordnung von jeweils 8 Bytes in einem Block untereinander. Von diesen Blocks bilden dann 40 eine Zeile.

Diese Anordnung entspricht übrigens in etwa dem Aufbau des Zeichengenerators, in dem die Zeichen auch immer in 8-Byte-Blöcken abgelegt sind.

Auf Grund dieser Grafikverwaltung wird ein Punkt dann dadurch gesetzt, daß das korrespondierende Bit im Speicher auf 1 gesetzt wird. Das folgende Schaubild soll Ihnen noch einmal den Aufbau des Grafikspeichers bildlich demonstrieren:



Um nun jeden Punkt genau zu definieren, teilt man das Grafikfenster in 25 Zeilen (0-24) mit jeweils 40 Spalten (0-39) auf. Jede dieser Spalten besteht für sich wieder aus 8 Bits (0-7), den sogenannten senkrechten Reihen. Jede Zeile wiederum besteht ebenfalls aus 8 übereinander angeordneten waagerechten Reihen. Der Grafikschirm kann also in 40*8 senkrechte Reihen (0-319) und 25*8 waagerechte Reihen (0-199) aufgeteilt werden. Dies ergibt eine Auflösung von 320*200 Punkten.

Um nun einen Grafikpunkt eindeutig zu bestimmen, könnte man sagen: Punkt X ist das Yte Bit im Byte Z. Diese Einteilung wäre allerdings kompliziert und unübersichtlich. Daher definiert man die Koordinaten eines Punktes wie folgt:

Die X-Koordinate wird mit der Nummer der senkrechten Reihe und die Y-Koordinate mit der Nummer der waagerechten Reihe gleichgesetzt. Die Koordinate 111/49 wäre also der Schnittpunkt der 111ten senkrechten Reihe mit der 49ten waagerechten Reihe. Oder noch genauer das 0te Bit des 13ten 8*8-Blocks in der 0ten waagerechten Reihe der 6ten Zeile.

Die Aufteilung des Bildschirms in 319 mögliche X-Koordinaten und 199 mögliche Y-Koordinaten entspricht übrigens in etwa dem Koordinatensystem der Mathematik. Hier jedoch liegt der Koordinatenursprung in der linken oberen, statt in der linken unteren Ecke. Somit werden dann die Y-Koordinaten von oben nach unten aufsteigend gezählt. Um also den Koordinatennullpunkt auch im Grafikspeicher nach unten links zu legen, müßte man die Y-Koordinate durch folgende Formel umrechnen:

Yn = Maximale Anzahl der waagerechten Reihen minus der Y-Koordinate

also: Yn = 199 - Y

Doch Obriges nur für die Interessierten. Viel wichtiger für uns sollte sein, wie wir die X- und Y-Koordinaten wieder in das Format des Grafikspeichers umrechnen. Denn die Aufteilung in 319 X- und 199 Y-Koordinaten ist eine Vereinbarung unsererseits. Der Computer versteht diese Angaben nicht. Also müssen wir den Punkt wieder durch das Xte Bit im Yten Byte für den Computer verständlich angeben.

Um die Zeile (0-25) in der sich der Punkt befindet, zu ermitteln, müssen wir die Y-Koordinate lediglich ohne Rest durch 8 dividieren.

Zeile = INT(Y/8)

Der Vorkommateil des Quotienten Y/8 gibt dann die Zeile an, während über dem Nachkommateil die horizontale Reihe bestimmt werden kann. Da jede Zeile aus 40 nebeneinander liegenden Acht-Byte-Päckchen, also 320 Bytes, besteht, müssen wir die weiter oben ermittelte Y-Adresse mit 320 multiplizieren:

Relative Adresse der horizontalen Reihe = INT(Y/8)*320

Die untersten drei Bits (Divisionsrest) des Y-Wertes geben nun die Reihe in der Zeile an und müssen auch noch zuaddiert werden. Unsere Formel zur Berechnung der horizontalen Reihe sieht daher jetzt so aus:

Adresse der horizontale Reihe = INT(Y/8)*320+(Y AND 7)

Nun wissen wir nur noch nicht, welches Bit in dem wievielten Byte der Reihe wir ansprechen müssen. Die Anzahl der Bytes gerechnet vom Reihenstart errechnet sich so:

Bytenummer = INT(X/8)*8 oder Bytenummer = X AND 504

Wir mußten zunächst durch 8 teilen, da ein Byte 8 Punkte darstellt. Dann jedoch multiplizieren wir das Ganze (ohne Rest) wieder mit 8, da jeweils 8 Bytes hintereinander einen 8*8-Block bilden. Durch die bis jetzt erhaltenen Formeln können wir wie folgt eindeutig die Adresse des Bytes ermitteln, in dem sich der gesuchte Punkt befindet:

Adr. = Grafikadr. + INT (Y/8) * 320+(Y AND 7) + X AND 504

Die Nummer des jeweiligen Bits errechnen wir, wie auch schon die horizontale Reihe, durch die 'AND' Funktion.

Also: Bitnummer = 7-(X AND 7)

Diesen Wert benutzen wir jetzt als Exponent zur Basis 2, um in BASIC den Wert dieses Bits zu berechnen, den wir dann direkt als Maske zum Setzen eines Punktes verwenden können.

```
Wert = 2^(7-(X AND 7))
```

Jetzt kennen wir die beiden Formeln zur Errechnung der Adresse und der Maske eines Punktes. Setzen wir also nun einen Punkt in der HRG durch das folgende kleine Programm, in dem wir unsere Grafik durch einen kleinen Trick mittels dem DIM-Befehl löschen.

Programmlisting:

```
100 rem ********************
110 rem *   punkt setzen   *
120 rem ********************
130 :
140 dim a$(10000) : clr : rem grafikspeicher löschen
150 :
160 v = 53248 : rem basisadresse vic
170 :
180 print chr$ (147);
190 :
200 poke v+17,peek (v+17) or 32 : rem grafik an
210 poke v+24,24 : rem grafikspeicher bei 8192
220 :
230 for i=1024 to 2023 : poke i,7+0*16 : next : rem farbe
setzen
235 :
240 rem ************************
250 rem *   punkt berechnen   *
```

```
260 rem ************************
265 :
270 x=111 : y=49 : bs=8192 : rem x/y koordinaten, basis
adresse
280 :
290 ad=bs+int (y/8)*320+(y and 7)+int (x/8)*8
300 mk=2^(7-(x and 7))
310 :
320 poke ad,peek (ad) or mk : rem punkt hinzu 'or'en
330 :
340 wait 198,1 : poke 198,0
350 :
360 poke v+17,peek (v+17) and not 32 : rem grafik aus
370 poke v+24,21
380 :
390 end
```

Natürlich ist diese Berechnung in BASIC äußerst langsam. Deshalb haben wir im folgenden für Sie ein kleines Grafikpaket abgedruckt, das Ihnen Einschalten und Löschen der Grafik, sowie Punkt setzen und löschen und Ausschalten der Grafik abnimmt:

2.4.2 Ein kleines Grafikpaket

Hier ist nun das versprochene Grafikpaket. Einigen wird dies sicherlich bekannt vorkommen. Der Autor dieses Grafikpaketes, Axel Plenge, den Sie sicherlich schon kennen, war so freundlich, es uns zur Verfügung zu stellen. Zugegeben, dieses BASIC-Lader-Listing ist ziemlich lang, jedoch lohnt sich das Abtippen, gemäß an dem, was es zu leisten vermag, bestimmt.

Aus Gründen der Platzersparnis haben wir es vermieden, das ganze Assemblerlisting mit abzudrucken. Sollten Sie näher an Grafik interessiert sein, so möchte ich Sie auf das "Grafikbuch zum Commodore 64" aufmerksam machen, das übrigens auch aus der Feder Axel Plenges stammt.

Doch nun zu unserem Grafikpaket:

```
100 for i = 51200 to 52054
110 read x : poke i,x : s=s+x : next
120 data  76, 36,200, 76, 65,200, 76, 18,201, 76, 49,201
130 data  76, 42,201, 76, 88,200, 76, 85,200, 76,107,200
140 data  76,104,200, 76, 67,202, 76, 82,202, 76,105,202
150 data 234,173, 17,208,141, 30,203,173, 24,208,141, 31
160 data 203,169, 59,141, 17,208,169, 24,141, 24,208,169
170 data  96,141, 36,200, 96,173, 30,203,141, 17,208,173
180 data  31,203,141, 24,208,169,234,141, 36,200, 76, 68
190 data 229,162,  0, 44,162,128,134,151, 32,253,174, 32
200 data 121,200, 32,148,200, 76,226,200,162,  0, 44,162
210 data 128, 32, 90,200, 32,253,174, 32,121,200, 76,187
220 data 201, 32,235,183,138,168,166, 21,192,200,176, 13
230 data 165, 20,224,  1,144,  6,208,  5,201, 64,176,  1
240 data  96, 76, 72,178,140, 28,203,141, 26,203,142, 27
250 data 203,133, 20,134, 21,152, 74, 74, 74,170,189, 47
260 data 203,133,173,138, 41,  3,170,189, 73,203,133,172
270 data 152, 41,  7, 24,101,172,133,172,165, 20, 41,248
280 data 133, 99,169, 32,  5,173,133,173, 24,165,172,101
290 data  99,133,172,165,173,101, 21,133,173,165, 20, 41
300 data   7, 73,  7,170,189, 77,203,133,171, 96,160,  0
310 data   8,165,171, 36,151, 48,  5, 73,255, 49,172, 44
320 data  17,172,145,172,165,172,133,253,165,173, 74,102
330 data 253, 74,102,253, 74,102,253, 41,  3,  9,  4,133
340 data 254,173, 29,203,145,253, 40,164,111, 96,169, 32
350 data 133,254,160,  0,132,253,162, 32,152,145,253,200
360 data 208,251,230,254,202,208,246, 76, 55,201, 32,241
370 data 183,142, 29,203, 96, 32,241,183,142, 29,203,162
380 data   3,169,  4,133,254,160,  0,132,253,132,151,173
390 data  29,203,145,253,200,196,151,208,249,230,254,202
400 data 240,  3, 16,242, 96,162,232,134,151,208,235,165
410 data 172, 41,  7,201,  7,240,  5, 56,169,  0,176,  4
420 data 169, 56,230,173,101,172,133,172,169,  0,101,173
430 data 133,173, 96, 48,226,165,172, 41,  7,240,  5, 24
440 data 169,255,144,  4,169,199,198,173,101,172,133,172
450 data 165,173,233,  0,133,173, 96, 70,171,144, 14,102
460 data 171,165,172,200, 24,105,  8,133,172,144,  2,230
```

```
470 data 173, 96, 16,235,  6,171,144,  14, 38,171,165,172
480 data 136, 56,233,  8,133,172,176,  2,198,173, 96, 72
490 data 173, 27,203, 74,173, 26,203,106, 74, 74,133,111
500 data 104, 72, 56,237, 26,203, 72,138,237, 27,203,133
510 data 108,176, 10,104, 73,255,105,  1, 72,169,  0,229
520 data 108,133,106,133,110,104,133,105,133,109,104,141
530 data  26,203,142, 27,203,152, 24,237, 28,203,144,  4
540 data  73,255,105,254,133,107,140, 28,203,102,108, 56
550 data 229,105,170,169,255,229,106,133,112,164,111,176
560 data   5, 10, 32,166,201, 56,165,109,101,107,133,109
570 data 165,110,233,  0,133,110,132,111, 32,226,200,232
580 data 208,  5,230,112,208,  1, 96,165,108,176,222, 32
590 data 119,201, 24,165,109,101,105,133,109,165,110,101
600 data 106, 80,221, 32,253,174, 32,212,225,160, 32,162
610 data   0,169,  0, 76,213,255, 32,253,174, 32,212,225
620 data 162, 63,160, 64,169,  0,133,253,169, 32,133,254
630 data 169,253, 76,216,255, 32,241,183,134,103, 32, 15
640 data 243, 32, 31,243,166,103, 32,201,255,169,255,133
650 data  97,169,  7,133,253,169, 28,133,151,169,  0,141
660 data  32,203,169, 40,141, 33,203,162,  4,189, 42,203
670 data  32,210,255,202, 16,247,169,  0,133, 99,133,100
680 data 173, 32,203,133,101,169,  0,133,254,165, 99,166
690 data 100,164,101, 32,148,200,160,  0,177,172,166,254
700 data 157, 34,203,230,101,232,134,254,228,253,208,229
710 data 169,  0,160,  7,166,253, 30, 34,203, 42,202, 16
720 data 249, 37, 97,  9,128, 32,210,255,136, 16,237,165
730 data  99, 24,105,  8,133, 99,144,  2,230,100,206, 33
740 data 203,208,181,169, 13, 32,210,255,173, 32,203, 24
750 data 105,  7,141, 32,203,198,151,240,  3, 76,138,202
760 data 169,  4,197,253,240, 12,133,253,169,  1,133,151
770 data 169, 15,133, 97,208,235,169, 15, 32,210,255, 76
780 data 204,255, 48, 48, 48, 32, 58, 32,130, 32, 88, 32
790 data  58, 32,143, 32, 87, 65, 80,  0, 16, 27,  8,  0
800 data   1,  2,  3,  5,  6,  7,  8, 10, 11, 12, 13, 15
810 data  16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 30
820 data  31,  0, 64,128,192,  1,  2,  4,  8, 16, 32, 64
830 data 128, 76, 79
840 if s <> 104883 then print "fehler in datas !!" : end
850 print "ok"
```

Funktionserklärung:

```
Einschalten der Grafik.............SYS 51200
Ausschalten der Grafik.............SYS 51203
Löschen der Grafik.................SYS 51206
Farbe setzen.......................SYS 51209,pf*16+hf
Plotfarbe ändern...................SYS 51212,pf*16+hf
Punkt setzen.......................SYS 51215,x,y
Punkt löschen......................SYS 51218,x,y
Linie zeichnen.....................SYS 51221,x1,y1,x2,y2
Linie löschen......................SYS 51224,x1,y1,x2,y2
Grafik laden.......................SYS 51224,"name",ga
Grafik speichern...................SYS 51230,"name",ga
Hardcopy auf Seikosha 100VC.........SYS 51233,lf
```

Dabei sind :

```
pf........Punktfarbe
hf........Hintergrundfarbe
x.........X-Koordinate eines Punktes
y.........Y-Koordinate eines Punktes
x2/x2.....X-Koordinate des Start- und Endpunktes
y1/y2.....Y-Koordinate des Start- und Endpunktes
"name"....Filename
ga........Geräteadresse
lf........logische Filenummer
```

Bei der Hardcopy muß Folgendes beachtet werden: Grundsätzlich muß dem Hardcopybefehl ein "OPEN lf,ga" vorausgehen. Danach erst kann die Hardcopy gedruckt werden.

Am besten lernen Sie im folgenden Kapitel mit den neuen Befehl umzugehen.

2.4.3 3-D-Graphik

Wir wollen Ihnen nun ein BASIC-Programm vorstellen, das mit Hilfe der hochauflösenden Farbgrafik des Commodore 64 dreidimensionale Darstellungen auf den Bildschirm bringt. Das Programm benutzt die Befehle des Grafikpaketes. Im Anschluß finden Sie jedoch die erforderlichen Änderungen, falls Sie dieses Programm nicht verwenden wollen.

Das Programm stellt eine Funktion dar, die Sie in Zeile 100 des Programms definieren können. Die Funktion kann in drei verschiedenen Arten dargestellt werden:

Erstens in einem normalen kartesischen (rechtwinkligen) Koordinatensystem, so wie Sie die Funktion auch auf Millimeterpapier zeichnen würden.

Zum zweiten ist die Darstellung in Polarkoordinaten (Angabe von Radius und Winkel) möglich.

Am interessantesten ist zweifellos die dreidimensionale Darstellung. Die Funktion wird dazu um die (senkrechte) Y-Achse gedreht. Wegen der Vielzahl der zu berechnenden Punkte braucht diese Art der Darstellung jedoch die längste Zeit. Die Parameter der Darstellung können Sie weitestgehend beeinflussen.

Nun zur Beschreibung des Programms. Zuerst können Sie die Darstellungsart auswählen (Zeile 40-70). Bei der kartesischen und Polardarstellung werden Sie dann nach dem Funktionsinkrement gefragt (Zeile 260). Das ist der Wert, um den der Parameter der waagerechten Achse nach jedem berechneten Punkt erhöht wird. Zeile 270 und 280 fragen nach dem Faktor für X- und Y-Achse. Damit können Sie das Maßstabsverhältnis der Achsen bestimmen. Für den Anfang können Sie beide Male 1 eingeben. Bei der horizontalen und vertikalen Verschiebung (Zeile 370-410) können wir vorerst je Null eingeben. In Zeile 430 wird auf den Graphikmodus

umgeschaltet. Die Zeilen 450 bis 560 zeichnen das Achsenkreuz und die Skalierung. Die Zeilen 680 bis 790 zeichnen die Polardarstellung, von Zeile 820 bis 970 steht die rechtwinklige Darstellung.

Ab Zeile 1010 steht die dreidimensionale Darstellung. Zuerst können Sie wieder Maßstab und Lage der Darstellung bestimmen; geben Sie für den Anfang die vorgeschlagenen Werte 20 bzw. 90 ein. Für die dreidimensionale Darstellung muß die Funktion in Zeile 100 mehrere zehntausend Male berechnet werden; das Programm benötigt dazu je nach Funktion ca. eine halbe bis mehrere Stunden.

Lassen Sie das Programm einmal mit verschiedenen Funktionen laufen; als Anregung hier ein paar Funktionen, die interessante Darstellungen ergeben.

```
100 DEF FN R(Q) = COS (2*Q) + COS ((Q + BB)/16)
100 DEF FN R(Q) = SQR (ABS(.5*(16-Q*Q)) + 1/(Q+4)
100 DEF FN R(Q) = COS (4*Q) + 20/(Q*Q + 3)
```

Im folgenden nun das Listing:

```
1 GI = 51200 : GA = 51203 : GC = 51206
2 SP = 51215 : CP = 51218 : SC = 51209
3 SL = 51221 : CL = 51224
4 POKE 53280,0:POKE53281,0
5 :
10 PRINT"■■  GRAFISCHE DARSTELLUNG VON FUN
KTIONEN■■"
20 PRINT"  DEFINIERT IN ZEILE 100■■"
40 PRINT"■■  1 - KARTESISCHE DARSTELLUNG"
50 PRINT"■■  2 - POLAR-KOORDINATEN"
60 PRINT"■■  3 - 3D-DARSTELLUNG"
70 INPUT"■■  AUSWAHL: 1■■■■";PL
100 DEF FNR(Q)=COS(Q)+COS(2*Q)+COS(5*Q)
210 IFPL=3THEN1010
250 PRINT:PRINT
```

```
260 INPUT"FUNKTIONS-INKREMENT =";IK
270 INPUT"■FAKTOR FUER X-ACHSE =";S1
280 INPUT"■FAKTOR FUER Y-ACHSE =";S2
370 PRINT"■URSPRUNG LINKS ODER RECHTS VERS
CHIEBEN"
380 INPUT"ZAHL VON -130 BIS 130 ";C
400 PRINT"■URSPRUNG HOCH ODER RUNTER VERSC
HIEBEN"
410 INPUT"ZAHL VON  -90 BIS  90 ";D
430 SYS GI:SYS GC:SYS SC,5*16
450 E=100+D:F=160+C
470 SYS SL,F,0,F,199
480 SYS SL,0,E,319,E
490 FORXR=FTO319STEP19*S1
500 SYS SL,XR,E-1,XR,E+1
510 FORXL=FTO0STEP-19*S1
520 SYS SL,XL,E-1,XL,E+1
530 FORYD=ETO199STEP15*S2
540 SYS SL,F-1,YD,F+1,YD
550 FORYU=ETO0STEP-15*S2
560 SYS SL,F-1,YU,F+1,YU
580 IFPL=1THEN820
610 REM POLAR PLOT
690 RD=π/180:FORG=0TO360STEPIK:T=G*RD
710 X=FNR(T)*COS(T):Y=FNR(T)*SIN(T)
730 XX=X*(19*S1)+F:YY=-Y*(15*S2)+E
740 IFXX<0ORXX>319THEN780
750 IFYY<0ORYY>199THEN780
770 SYS SP,XX,YY
780 NEXT
790 SYS GA:END
820 REM CARTESIAN PLOT
830 FORG=0TO319STEPIK
840 X=(G-F)/(19*S1):Y=FNR(X)
850 YY=E-(Y*15*S2)
860 IFYY<0ORYY>199THEN960
900 SYS SP,G,YY
960 NEXT
970 SYS GA:END
1010 REM 3D-PLOT
```

```
1020 PRINT"■■■■■■■VERTIKAL-MASSTAB"
1030 PRINT"■■■■■■-40 BIS 40    20■■■■■■■";:INPU
TN1
1040 PRINT"■■■■■■■VERTIKALE LAGE"
1050 PRINT"■■■■■■-50 BIS 150   90■■■■■■■";:INPU
TN2
1260 REM KONSTANTEN A,B,C,D,E,F,G
1280 A=144:B=2.25:C=N1:D=.0327:E=160:F=N2:
G=199
1400 SYS GI:SYS GC:SYS SC,5*16
1410 FORH=-ATOASTEPB
1420 AA=INT(.5+SQR(A*A-H*H))
1430 FORBB=-AATOAA:CC=SQR(BB*BB+H*H)*D
1440 D1=FNR(CC):DD=D1*C:GOSUB1520:NEXT:NEX
T:SYS GA:END
1450 GOTO1450
1520 X=BB+H/B+E:Y=DD-H/B+F
1530 X1=INT(.85*X):Y1=INT(.9*(G-Y)):IFY<00
RY>199THENRETURN
1600 SYS SP,X1,Y1
1620 SYS CL,X1,Y1+1,X1,199:RETURN
```

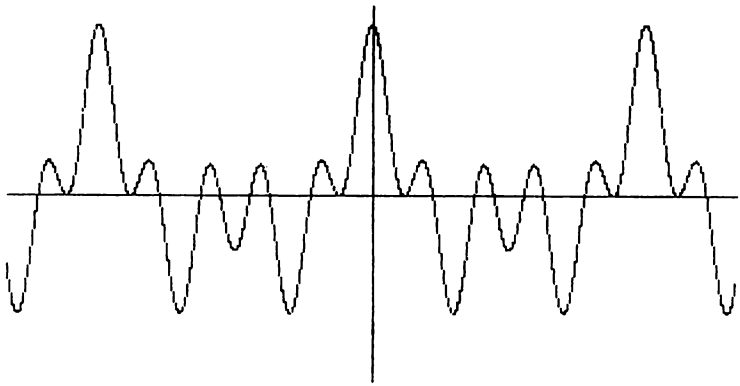
Verwenden Sie das Grafikpaket nicht, müssen Sie folgende Änderungen und Ergänzungen im Programm durchführen:

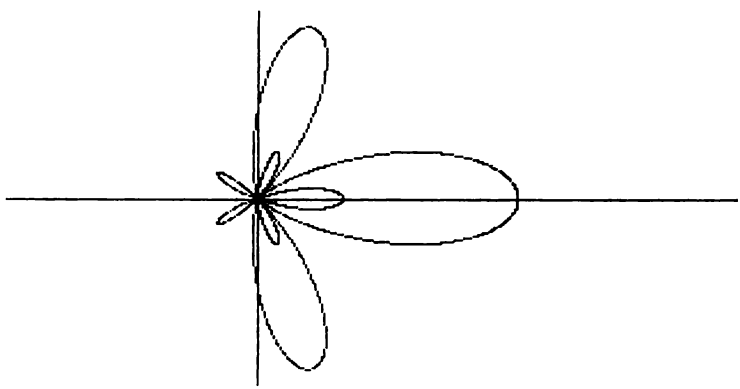
```
Zeile 5  POKE 56,32 : CLR
Zeile 430 und 1400: Jeweils GOSUB 2000
Zeile 470  FOR A1=0 TO 199 : AX=F: AY=A1: GOSUB 3000 : NEXT
Zeile 480  FOR A1=0 TO 319 : AX=A1: AY=E: GOSUB 3000 : NEXT
Zeile 500  FOR A1=E-1 TO E+1: AX=XR: AY=A1: GOSUB 3000: NEXT
Zeile 520  FOR A1=E-1 TO E+1: AX=XL: AY=A1: GOSUB 3000: NEXT
Zeile 540  FOR A1=F-1 TO F+1: AX=A1: AY=YD: GOSUB 3000: NEXT
Zeile 560  FOR A1=F-1 TO F+1: AX=A1: AY=YU: GOSUB 3000: NEXT
Zeile 770  AX=XX: AY=YY: GOSUB 3000
Zeile 900  AX=G: AY=YY: GOSUB 3000
Zeile 1600 AX=X1: AY=Y1: GOSUB 3000
```

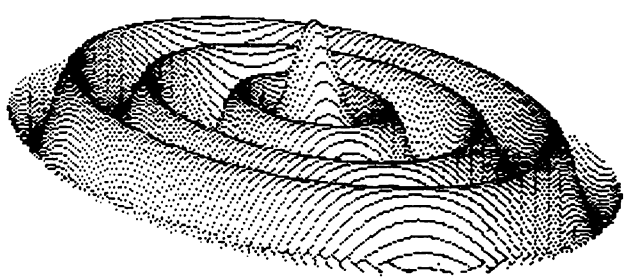
```
Zeile 1620 GOSUB 4000 : RETURN
Zeile 2000 FOR A1=8192 TO 16191 : POKE A1,0 : NEXT
Zeile 2010 FOR A1=1024.TO 2023  : POKE A1,16: NEXT
Zeile 2020 POKE 53248+17, 27+32 : POKE 53248+24, 16+8
Zeile 2030 RETURN
Zeile 3000 OY=320*INT(AY/8)+(AYAND7)
Zeile 3010 OX=8*INT(AX/8)
Zeile 3020 MA=2^((7-AX)AND7)
Zeile 3030 AV=8192+OY+OX
Zeile 3040 POKE AV,PEEK(AV) OR MA : RETURN
Zeile 4000 FOR A1=Y1+1 TO 199: AX=X1: AY=A1: GOSUB 5000: RETURN
Zeile 5000 OY=320*INT(AY/8)+(AYAND7)
Zeile 5010 OX=8*INT(AX/8)
Zeile 5020 MA=2^((7-AX)AND7)
Zeile 5030 AV=8192+OY+OX
Zeile 5040 POKE AV,PEEK(AV) AND NOT MA : RETURN
```

Durch die Programmierung der Grafikfunktionen in BASIC wird das Programm natürlich bedeutend langsamer als wenn Sie mit Hilfe des Grafikpaketes programmieren.

Auf der nächsten Seite finden Sie die Ergebnisse dieses Programms.







2.5 Zeichensatzmodifikation

Eine der Besonderheiten des Commodore 64 ist die Möglichkeit, den Zeichengenerator, der natürlich im ROM, also im Festwertspeicher untergebracht ist, in das RAM zu legen. Dort können Sie dann die entsprechenden nach Belieben verändern und so Ihren eigenen Zeichensatz definieren.

Der Speicherbereich, in dem das Aussehen der einzelnen Zeichen festgehalten wird, nennt sich Charaktergenerator oder kurz Chargenerator. Dieses ROM liegt im Bereich von 54348 bis 57343 ($D000-$DFFF). Leider ist dieser Bereich von BASIC aus nur recht schwer zugänglich, da sich dort die folgenden drei Speicherbereiche überlappen:

1) Die Register des VIC, der I/O-Ports, und des SID
2) Der Charaktergenerator
3) Ein Teil des RAMs

Nun gibt es aber eine bestimmte Speicherstelle, die für die Speicherverwaltung zuständig ist. Es ist dies die Speicherstelle 1, bei der die Bits 0-2 für die Speicherverwaltung zuständig sind. Sie besagen, welche Art des Speichers an den verschiedenen Stellen erscheinen soll, so auch, wann der Chargenerator ausgelesen werden kann und wann nicht. Für den Chargenerator ist Bit 2 in der Speicherstelle 1 zuständig. Leider kann dieses Bit nicht einfach gelöscht werden, um so den Chargenerator lesbar zu machen, da eine bestimmte Routine, die Interruptroutine, 60 mal pro Sekunde aufgerufen wird, und dort die Registern der I/O-Bausteine manipuliert, die dort nun nicht mehr verfügbar wären. Dem würde dann ein völliger Systemabsturz folgen.

Wir können dem aber vorbeugen, indem wir uns eines kleinen Tricks bedienen. Es besteht selbst in BASIC die Möglichkeit, durch Ausschalten der Interruptquelle, das ständige Anspringen der Interruptroutine, die übrigens den Cursor blinken läßt, und auch für die Tastaturabfrage zuständig ist, einen Absturz des Systems zu vermeiden.

2.5.1 Verschieben des Zeichensatzes

Im folgenden Programm können Sie sich nun ein Zeichen Ihrer Wahl in vergrößerter Form auf dem Bildschirm anzeigen lassen. Gleichzeitig erlaubt Ihnen das Programm, das Prinzip der Chargeneratorprogrammierung zu erkennen:

```
100 rem ***********************************
110 rem *** auslesen des chargenerators ***
120 rem ***********************************
130 print chr$(147);140 input "bitte ein zeichen
eingeben";a$
150 print chr$(147)+a$:c=peek(1024)
160 cg=13*4096 : rem start des chargenerators
170 b=(peek(53248+24)and2)*1024
180 for i=0 to 7
190 :poke56320+14,0:poke1,51
200 :z=peek(cg+b+8*c+i)
210 :poke1,55:poke56320+14,1
220 :for j=7 to 0 step-1
230 ::a=z and 2^j
240 ::if a then print "*";:goto260
250 ::print ".";
260 :next j
270 :print
280 next i
```

Das Programm fragt zunächst nach dem Zeichen, dessen Matrix Sie ansehen wollen. Der Bildschirmcode des Zeichens wird in Zeile 150 in die Variable c geholt. Zeile 140 prüft auf Klein/Groß oder Groß/Grafikmodus. Danach wird in Zeile 160 die Startposition der Zeichenmatrix im Chargenerator bestimmt. Die folgenden Zeilen lesen nun jeweils ein Byte aus dem Chargenerator aus und bestimmen in dessen Abhängigkeit, ob ein Punkt gesetzt werden muß oder nicht.

Geben Sie z. B. ein "T" ein, so erhalten Sie folgende Ausgabe:

```
.******.
...**...
...**...
...**...
...**...
...**...
...**...
........
```

Nachdem Sie sich nun die Matrix einzelner Zeichen angesehen haben, können wir daran gehen eigene Zeichen zu definieren. Dazu müssen wir den Chargenerator vom ROM ins RAM übertragen. Nun stehen uns aber verschiedene Speicherbereiche zur Verfügung. Wir haben nun die Aufgabe, uns für den Besten zu entscheiden. Zuvor muß noch gesagt werden, daß der Videocontroller nur 16K auf einmal adressieren kann. D. h., daß es uns nicht möglich ist, den Chargenerator nach $C000 zu legen, während der Bildschirmspeicher bei $400 (1024) liegt. Aber der Chargenerator liegt ja nach dem Einschalten auch bei $D000, während der Bildschirmspeicher bei $400 liegt. Im Prinzip stimmt das, nur daß in diesem Falle der Bereich von $D000-$DFFF hardwaremäßig nach $1000-$1FFF "gespiegelt" wird. Daher wird dieser Bereich auch "verbotener Bereich" genannt, da in diesem Speicherabschnitt keine Grafiken oder Zeichensätze liegen dürfen.

Da der VIC selbständig nur 16K adressieren kann, kann der gesamte VIC-Bereich durch die Bits 0 und 1 der CIA II bestimmt werden. Aber Vorsicht! - Diese beiden Bits sind Low-Aktiv, d. h. es gilt ein Bit gesetzt, wenn es gelöscht ist, und umgekehrt.

Doch nun weiter im Text. Wir haben genug Informationen, um unseren Chargenerator in das RAM zu kopieren. Das kann von

BASIC aus mit einer kleinen POKE-Schleife geschehen. Anschließend teilen wir dem neuen Betriebssystem noch die Lage des neuen Bildschirmspeichers mit, der ja zwangsweise mit verschoben werden muß:

```
100 for i=13*4096 to 14*4096-1
110 poke 56320+14,0 : poke 1,51
120 poke i+4096,peek(i)
130 poke 1,55 : poke 56320+14,1
140 poke 53272,24 : rem Lage der Screen dem VIC mitteilen
150 poke 648,196 : rem Lage der Screen dem Btrsys.
mitteilen
160 poke 56576,148 : rem 16K Bereich auswählen
```

2.5.2 Verändern des Zeichensatzes

Für bestimmte Zwecke ist es oft wünschenswert, besondere Zeichen und Symbole zur Verfügung zu haben, die auf Tastendruck sofort erscheinen. Darunter fallen z.B. griechische Buchstaben, die in mathematischen Formeln häufig benutzt werden.

Denkbar sind auch kleine Spielfiguren, die zwar auch mit Sprites erzeugt werden können, was allerdings etwas aufwendiger sowohl beim Erstellen als auch in der Handhabung ist.

Wenn Ihnen die geeigneten Verfahren geläufig sind, können Sie sich Ihre Figuren zunächst als Rasterdarstellung auf ein Blatt Papier malen und dann die ermittelten Werte in das Duplikat des Zeichengenerators poken, aber das ist bestimmt nicht jedermanns Sache.

Wir haben daher ein kleines Programm geschrieben, mit Hilfe dessen Sie das gewünschte Zeichen bequem am Bildschirm entstehen lassen und seine Entwicklung verfolgen können.

Voraussetzung hierzu ist lediglich ein Joystick am Controlport 1.

Die Wirkungsweise des Programmes ist die, daß zunächst zwei Duplikate des Zeichengenerators ins RAM kopiert werden.

Aus dem ersten Duplikat wird ein Zeichen entnommen und als Sprite dargestellt, und zwar einmal in Originalgröße und einmal in doppelter Größe, der besseren Lesbarkeit wegen.

In den Sprites erscheint ein blinkender Punkt, der sog. Mikrocursor, den Sie nun mit Hilfe des Joysticks bewegen können.

Die gewünschte Aktion, nämlich Linie ziehen, Linie löschen oder nur positionieren, können Sie durch Druck auf den Knopf am Joystick auswählen.

Ist das Zeichen zu Ihrer Zufriedenheit ausgefallen, so drücken Sie die Taste F1.

Das neue Zeichen wird nun an die angezeigte Position des Alternativzeichensatzes (2. Duplikat) übernommen.

Soll das dargestellte Zeichen unverändert bleiben, drücken Sie die Taste G.

Nun können Sie das nächstfolgende Zeichen bearbeiten.
Nachdem Sie alle 512 Zeichen bearbeitet haben, beendet sich das Programm.

Wieso eigentlich 512 Zeichen?

Nun, für eine Betriebsart, nämlich Groß/Grafik oder Groß/Kleinschrift, gibt es 128 darstellbare Zeichen. Dieselben Zeichen noch einmal in reverser Darstellung macht schon 256. Das ergibt für beide Betriebsarten zusammen 512.

Die den Zeichen zugeordneten Positionen im Zeichengenerator können Sie den Seiten 132ff im Bedienungshandbuch des C64 entnehmen.

Bevor wir das Programm im einzelnen erörtern, sollten Sie die Bedeutung der verwendeten Variablen und Speicheradressen kennen.

Zunächst die Variablen:

C	Basisadresse des 1. Duplikats des als nächstes anzuzeigenden Zeichens.
CD	Basisadresse des zu übertragenden Zeichens im 2.Duplikat
CP	Zeichenpositionszähler im Bereich 0-511
JB	Zustand des Knopfes am Joystick
JR	Knüppelstellung des Joystick
JS	Adresse des Controlport 1
MA	Zähler für den Betriebszustand
PO	Mikrocursorposition innerhalb des adressierten Bytes
PP	Adresse innerhalb der Spritedaten, an der sich gerade der Mikrocursor befindet
PV	Augenblickswert des Bytes der Spritedaten, das durch den Joystick gerade adressiert ist
SB	Basisadresse der Spritedaten
V	Basisadresse des Videocontrollers
X	x-Position der Sprites auf dem Bildschirm
XJ	x-Position des durch den Joystick gesteuerten Mikrocursors
Y	y-Position der Sprites auf dem Bildschirm
YJ	y-Position des durch den Joystick gesteuerten Mikrocursors

Nun die Adressen:

56	Höherwertiges Byte des Pointer auf die Speicherobergrenze
648	Pointer auf den Beginn des Video-RAM
832	Startadresse des Kassettenpuffers. Da dieser innerhalb des Programmes nicht anderweitig benötigt wird, ist hier das Maschinenprogramm hinterlegt.
50169	Pointer für Sprite 1
50170	Pointer für Sprite 2
53272	Pointer für Video-RAM und Zeichengenerator innerhalb des Videocontroller
56576	Hier stehen die beiden Bits zur Festlegung des 16K-Bereichs für die mit dem Bildschirm zusammenhängenden Speicher.

Es folgt die schrittweise Einführung in das Programm:

10 Die Speicherobergrenze wird herabgesetzt, da hier das erste Duplikat des Zeichensatzes hinterlegt werden soll.

30-70 Die Sprites 1+2 werden eingeschaltet und deren Farbe gesetzt.

Nun werden die Spritepointer geladen und dann Sprite 2 auf doppelt große Darstellung geschaltet.

Die Spritedaten werden zunächst gelöscht und dann die Sprites ungefähr in der Bildschirmmitte positioniert.

1000-1010 Hier steht das Maschinenprogramm.

Für den Interessierten nachstehend die Assemblerschreibweise:

```
sei             ; Interrupt verhindern
lda #$33        ; Zeichengenerator zu-
sta 1           ; gänglich machen
lda #0          ;
sta $5f         ; alter Blockanfang low
sta $5a         ; altes Blockende low
sta $58         ; neues Blockende low
lda #$d0        ;
sta $60         ; alter Blockanfang high
lda #$f0        ;
sta $59         ; neues Blockende high
lda #$e0        ;
sta $5b         ; altes Blockende high
jsr $a3bf       ; Blockverschieberoutine
lda #$37        ; Alte Konfiguration wieder
sta #1          ; herstellen
cli             ; Interrupt erlauben
rts             ; zurück nach BASIC
```

1020-1040 Das Maschinenprogramm wird in den Kassettenpuffer gebracht und ausgeführt, das heißt von dem Zeichengenerator werden zwei Duplikate erzeugt.

1060 Dem Betriebssystem werden die veränderten Verhältnisse mitgeteilt.

Verändert wurde die Lage von Zeichengenerator und (wegen der Hardware zwangsläufig) Video-RAM. Die Zeichen, die Sie jetzt auf dem Bildschirm sehen, stammen bereits aus dem 2.Duplikat.

2000-2360 Die Zeichen werden nun nacheinander in die Spritedaten überführt und können verändert werden.

2380 Nach Behandlung aller Zeichen wird die Möglichkeit zum Abspeichern des neuen Zeichensatzes angeboten.

4000-4070 Der Joystick wird abgefragt und abhängig von der Stellung des Knüppels und des Knopfes verzweigt.

5000-12040 Abhängig von der Stellung des Joystick wird die Cursorposition aktualisiert.

13000-13200 Die Position des Joystick wird auf die Punkteposition des Mikrocursor umgerechnet und dieser Punkt abwechselnd an- und ausgeschaltet.

20000-20060 Hier wird das für gut befundene Zeichen in das 2. Duplikat übertragen.

30000-30200 Hier kann der neu erzeugte Zeichensatz auf Diskette (8) oder Kassette (1) abgespeichert werden.

Als Besonderheit dieses Programmes ist noch anzumerken, daß der modifizierte Zeichengenerator keinen BASIC-Speicher belegt. Er liegt nämlich im RAM unter dem Betriebssystem (das ist wörtlich zu nehmen, denn bekanntlich überlappen sich beim C64 RAM und ROM).

Das Video-RAM hat nun die Startadresse 49152, was Sie bitte berücksichtigen wollen, wenn Sie hineinpoken. Durch die Hardware des C64 ist es leider gegeben, daß bei Verlegen des Zeichengenerators auch eine Verschiebung des Video-RAMs stattfindet. Diese Problematik finden Sie im '64 intern' von DATA BECKER näher erläutert.

Abschließend das Programmlisting:

```
10 poke56,144:clr:v=53248
30 pokev+21,6:pokev+40,1:pokev+41,1
40 poke50169,16:poke50170,16:pokev+23,4:pokev+29,4
50 fori=0to62:poke50176+i,0:next
60 x=150:y=100:pokev+4,x:pokev+2,x-40:pokev+16,0
70 pokev+5,y:pokev+3,y+19
1000 data120,169,51,133,1,169,0,133,95,133,90,133,88,169,
```

```
208,133,96,169,240,133
1010 data89,169,224,133,91,32,191,163,169,55,133,1,88,96
1020 fori=832to832+33:reada:pokei,a:next
1040 sys832:poke850,160:sys832
1060 poke53272,8:poke56576,peek(56576)and252:poke648,192
1070 printchr$(147)
2000 c=9*4096
2020 forcp=0to511:printchr$(19)cp:sb=50176
2040 fori=0to7:pokesb+3*i,peek(c+i):nexti
2360 c=c+8:gosub4000:nextcp
2380 pokev+21,0:goto 30000
4000 xj=0:yj=0:js=56321:sb=50176:pp=832
4020 jr=(255-peek(js))and15:jb=(255-peek(js))and16
4030 ifjbthenma=ma+1:ifma>2thenma=0
4040 onjrgoto5000,6000,4020,7000,8000,9000,4020,10000,11000,12000
4045 ifpeek(203)<>4then4066
4050 printchr$(19)chr$(17)chr$(17)chr$(17)chr$(18)"save":gosub20000
4055 printchr$(19)chr$(17)chr$(17)chr$(17)chr$(17)"     ":return
4066 ifma=1thenprintchr$(19)chr$(17)chr$(17)chr$(18)" set"
4067 ifma=2thenprintchr$(19)chr$(17)chr$(17)chr$(18)" clr"
4068 ifpeek(203)=26thenreturn
4069 ifma=0thenprintchr$(19)chr$(17)chr$(17)"     "
4070 gosub13000:goto4020
5000 rem oben
5020 yj=yj-1:ifyj<0thenyj=0
5040 gosub13000:goto4020
6000 rem unten
6020 yj=yj+1:ifyj>7thenyj=7
6040 gosub13000:goto4020
7000 rem links
7020 xj=xj-1:ifxj<0thenxj=0
7040 gosub13000:goto4020
8000 rem links oben
8020 xj=xj-1:ifxj<0thenxj=0
8040 goto5000
9000 rem links unten
9020 xj=xj-1:ifxj<0thenxj=0
9040 goto6000
10000 rem rechts
10020 xj=xj+1:ifxj>7thenxj=7
```

```
10040 gosub13000:goto4020
11000 rem rechts oben
11020 xj=xj+1:ifxj>7thenxj=7
11040 goto5000
12000 rem rechts unten
12020 xj=xj+1:ifxj>7thenxj=7
12040 goto6000
13000 rem
13020 pp=sb+yj*3+int(xj/8):pv=peek(pp)
13040 po=xj-int(xj/8)*8
13060 ifpvand2^(7-po)thenpokepp,(pvand(255-2^(7-po))):goto13100
13080 pokepp,(pvor(2^(7-po)))
13100 ifma=1thenpv=(pvor(2^(7-po)))
13120 ifma=2thenpv=(pvand(255-2^(7-po)))
13200 fori=0to50:next:pokepp,pv:return
20000 rem neues zeichen uebertragen
20010 cd=c+20472
20020 fori=0to7
20040 pokecd+i,peek(sb+3*i):pokecd+i-20480,peek(sb+3*i)
20060 nexti:return
30000 input"abspeichern ";r$
30020 if r$<>"j"then 30200
30040 input"name ";n$
30060 input"geraeteadresse ";ad
30080 poke186,ad:poke781,0:poke782,160
30120 poke251,0:poke252,144:poke780,251
30140 poke183,len(n$):poke187,680and255:poke188,680/256
30160 fori=1tolen(n$)
30180 poke679+i,asc(mid$(n$,i,1)):next:sys65496
30200 poke56,160:clr
```

Das nachfolgende kleine Programm ermöglicht es, den Zeichensatz wieder einzulesen und zu aktivieren.

```
100 rem zeichensatz laden
110 input"name ";n$
120 input"geraeteadresse ";ad
130 poke186,ad:poke781,0:poke782,224
140 poke780,0:poke185,0:poke183,len(n$)
150 poke187,680and255:poke188,680/256
160 fori=1tolen(n$):poke679+i,asc(mid$(n$,i,1)):next
170 sys65493:rem load
180 poke53272,8:poke56576,peek(56576)and252:poke648,192
190 printchr$(147)
```

2.6 Die Sprites

Wie schon zu Beginn gesagt, besitzt der Commodore 64 nicht nur die Möglichkeit, einfache Texte oder Grafiken darzustellen, sondern ist auch in der Lage bis zu acht verschiedene (u.U. noch mehr, doch dazu später) Kleingrafiken, die sogenannten Sprites, darzustellen. Die Kleingrafiken werden ähnlich aufgebaut wie die HRG, jedoch besitzen sie nur eine Auflösung von 21*24 Punkten.

Natürlich muß wiederum einem Sprite ein bestimmter Speicherbereich zugewiesen werden. Dieser Bereich wird Block genannt. In diesem Block, der 64 Bytes umfaßt , wird das Aussehen des Sprites in Form eines Bit-Musters gespeichert.

Alle diese Informationen können Sie auch dem Commodore 64 Handbuch entnehmen. Leider aber sind die Informationen über Sprites sehr unvollständig, so daß ich im folgenden an den Beispielen im Commodore Handbuch anknüpfen und an diesen Beispielen die weiteren Möglichkeiten der Sprites erklären werde.

2.6.1 Kollision von Sprites

Es ist auch grundsätzlich möglich, Kollisionen von Sprites festzustellen. Hierzu dienen die Register 30 und 31, wobei Register 30 für eine Sprite/Sprite und Register 31 für eine Sprite/Hintergrund Kollision zuständig ist. Sollte ein Sprite mit einem Hintergrundzeichen, wobei es egal ist, ob dieses in der normalen Textgrafik oder in der hochauflösende Grafik passiert, zusammenstoßen, so werden die entsprechenden Bits im Register 30 gesetzt. Ebenso verfährt Register 31, nur mit dem Unterschied, daß es auf eine Sprite/Sprite-Kollision reagiert. Dort werden also immer 2 Bits auf einmal manipuliert.

Das folgende Programm soll Ihnen nun einmal eine Sprite/Hintergrund-Kollision demonstrieren. Dabei stütze ich mich wieder auf das Sprite, das im Commodore-Handbuch verwendet wird:

```
3 rem ************************************
4 rem ***                                ***
5 rem *** sprite/hintergrund kollision ***
6 rem *** -------------------------- ***
7 rem ***                                ***
8 rem ************************************
9 :
10 v=53248
20 for x=0 to 62
30 :read a:poke832+x,a
40 next x
50 poke 2040,13
60 poke v+21,peek(v+21)or2^0
70 :
80 print chr$(147);
90 for x=1 to 20
100 :print "*"
110 next
120 :
130 y=100:poke v+1,y
140 for x=255 to 0 step-1
```

```
150 :poke v,x
160 :if peek(v+31)=2^0 then 190
170 next x
180 :
190 rem ***collisions auswertung ***
200 :
210 for x=0 to 255
220 :poke 2040,x
230 next
240 pokev+21,0:end
250 :
260 rem *** sprite datas ***
270 :
280 data 0,127,0,1,255,192,3,255,224,3
290 data 231,224,7,217,240,7,223,240,7
300 data 217,240,3,231,224,3,255,224,3
310 data 255,224,2,255,160,1,127,64,1
320 data 62,64,0,156,128,0,156,128,0,73
330 data 0,0,73,0,0,62,0,0,62,0,0,62,0
340 data 0,28,0
```

2.6.2 Ein komfortabler Spriteeditor

Leider ist es immer noch recht schwer, Sprites zu editieren. Deshalb biete ich Ihnen im Folgenden einen Spriteeditor an, der zwar recht lang ist, mit dem Sie doch auf sehr einfache Weise Ihre Sprites editieren können.

Gleichzeitig will ich Sie an dieser Stelle auf die Diskette zum Buch verweisen, die für diejenigen unter Ihnen geeignet ist, die keine Zeit oder keine Lust oder auch beides nicht haben, die Programme in diesem Buch abzutippen.

Das dokumentierten Assemblerlisting finden Sie im Anhang, da dieses Listing an dieser Stelle den Lesefluß zu sehr stören würde. Doch nun der BASIC-Treiber mit BASIC-Lader des Assemblerprogramms:

```
10 REM ************************************
20 REM ***                              ***
30 REM ***    SPRITEGENERATOR V4.5      ***
40 REM ***   ------------------------   ***
50 REM ***                              ***
60 REM *** BASICTREIBER & BASICLADER    ***
70 REM ***                              ***
80 REM ************************************
90 :
100 POKE53280,0:POKE53281,0:POKE646,5:GOSU
B 60000
110 PRINT "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
"
120 PRINT "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
ALTE:   |"
130 PRINT "
————————┤"
140 PRINT "|                                |
        |"
150 PRINT "|                                |
        |"
160 PRINT "|                                |
        |"
170 PRINT "|                                |
"
180 PRINT "|                                |"
190 PRINT "|                                |"
200 PRINT "|                                |
"
210 PRINT "|                                |  |
|"
220 PRINT "|                                |  |
|"
230 PRINT "|                                |  |
|"
240 PRINT "|                                |
"
250 PRINT "|                                |
260 PRINT "|                                |
"
```

```
270 PRINT "|                                          |  |
    |"
280 PRINT "|                                          |  |
    |"
290 PRINT "|                                          |  |
    |"
300 PRINT "|                                          |  |
    |"
310 PRINT "|                                          |  |
    |"
320 PRINT "|                                          |  └───
"
330 PRINT "|                                          |
340 PRINT "|                                          |
350 PRINT "└──────────────────────────────────────────┘";
360 :
370 SG = 49152 : REM AUFRUF SPRITEGENERATO
R
380 RS = 51088 : REM BILDSCHRIM RETTEN
390 GS = 51111 : REM BILDSCHIRM HOLEN
400 CO = 51131 : REM WARMSTART DES SPRITEE
DITORS
410 :
420 SYS RS
430 PRINT "";
440 PRINT "
          ";
450 PRINT "                   M E N U E
          ";
460 PRINT "
          ";
470 PRINT "──────────────────────────────────────
──────────"
480 :
490 PRINT "SPRITE EDITIEREN...
....1"
500 PRINT "SPRITE SPEICHERN....
..2"
510 PRINT "SPRITE LADEN........
..3"
```

```
530 :
540 GET A$
550 A=VAL(A$):IF A<1 OR A>4 THEN 540
560 ON A GOTO 580,660,780
570 :
580 REM ********************************
590 REM *** SPRITE EDITIEREN **********
600 REM ********************************
610 :
620 POKE646,5:SYS CO:SYS GS:SYS SG
630 :
640 GOTO 430
650 :
660 REM ********************************
670 REM *** SPRITE ABSPEICHERN ********
680 REM ********************************
690 :
700 INPUT "□SPRITENAME :";N$
710 OPEN1,8,1,N$
720 FOR X=832 TO 832+64
730 :PRINT#1,CHR$(PEEK(X));
740 NEXT X:CLOSE 1
750 :
760 GOTO 430
770 :
780 REM ********************************
790 REM *** SPRITE LADEN **************
800 REM ********************************
810 :
820 INPUT "□SPRITENAME :";N$
830 OPEN1,8,0,N$
840 FOR X=57344+(PEEK(2)-1)*64 TO 57344+PE
EK(2)*64
```

```
850 :GET#1,A$:POKE X,ASC(A$+CHR$(0))
860 NEXT X:CLOSE 1
870 :
880 GOTO 430
890 :
60000 REM ********************************
60010 REM ***                          ***
60020 REM ***  BASIC LADER SPRITEGEN   ***
60030 REM *** ------------------------ ***
60040 REM ***                          ***
60050 REM ********************************
60060 :
60070 FOR X=49152 TO 51160
60080 :  READ A
60090 :  POKE X,A
60100 :  P=P+A
60110 NEXT X
60120 :
60130 IF P<> 268566 THEN PRINT "FEHLER IN
DATAS !!!":STOP
60140 :
60150 RETURN
60160 :
60170 DATA  32,101,195,169, 32,141,  0
60180 DATA 192,169, 98,141,  1,192,169
60190 DATA 199,141,  2,192, 76,254,195
60200 DATA 169,120,160,216,133,250,132
60210 DATA 251,165,247,162, 21,160, 24
60220 DATA 145,250,136,208,251, 72, 32
60230 DATA  49,192,104,202,208,241, 96
60240 DATA 165,250, 24,105, 40,144,  2
60250 DATA 230,251,133,250, 96,169,120
60260 DATA 160,  4,133,250,132,251,169
60270 DATA  32, 76, 31,192,169,120,160
60280 DATA   4,133,250,132,251,162, 21
60290 DATA 160, 24,177,250, 73,128,145
60300 DATA 250,136,208,247, 32, 49,192
60310 DATA 202,208,239, 76, 21,192, 32
60320 DATA 182,192,160, 25, 32,120,192
60330 DATA  32,165,192,160, 25, 76,120
```

```
60340 DATA 192,177,250,153,199,192,136
60350 DATA 208,248, 32,226,192, 32, 49
60360 DATA 192,162, 20,160, 24,177,250
60370 DATA 145,252,136,208,249, 32, 49
60380 DATA 192, 32,226,192,202,208,238
60390 DATA 160, 24,185,199,192,145,252
60400 DATA 136,208,248, 96,169,120,160
60410 DATA 216,133,250,132,251,169, 80
60420 DATA 160,216,133,252,132,253, 96
60430 DATA 169,120,160,  4,133,250,132
60440 DATA 251,169, 80,160,  4,133,252
60450 DATA 132,253, 96,  0,  0,  0,  0
60460 DATA   0,  0,  0,  0,  0,  0,  0
60470 DATA   0,  0,  0,  0,  0,  0,  0
60480 DATA   0,  0,  0,  0,  0,  0,  0
60490 DATA   0,  0,165,252, 24,105, 40
60500 DATA 144,  2,230,253,133,252, 96
60510 DATA 169, 54,133,  1, 32,250,192
60520 DATA 169, 55,133,  1, 96, 32, 74
60530 DATA 193, 32,107,193, 32, 49,192
60540 DATA  32, 18,193, 32,124,193, 32
60550 DATA  49,192, 32, 18,193, 76, 38
60560 DATA 193,162, 20,160, 24,177,252
60570 DATA 145,250,136,208,249, 32, 49
60580 DATA 192, 32,226,192,202,208,238
60590 DATA 160, 24,185, 32,163,153,120
60600 DATA   4,185, 32,168,153,120,216
60610 DATA 136,208,241, 96,169,160,160
60620 DATA   4,133,250,132,251, 96,169
60630 DATA 160,160,216,133,250,132,251
60640 DATA  96, 32,107,193, 32, 86,193
60650 DATA  32,124,193, 76, 86,193,162
60660 DATA  21,160, 24,177,250,145,252
60670 DATA 136,208,249, 32, 49,192, 32
60680 DATA 226,192,202,208,238, 96,169
60690 DATA 120,160,  4,133,250,132,251
60700 DATA 169,  0,160,160,133,252,132
60710 DATA 253, 96,169,120,160,216,133
60720 DATA 250,132,251,169,  0,160,165
```

```
60730 DATA 133,252,132,253, 96, 32,182
60740 DATA 192, 32,153,193, 32,165,192
60750 DATA  76,153,193,162, 21,160,  1
60760 DATA 177,250,141,200,192,200,177
60770 DATA 250,136,145,250,200,200,192
60780 DATA  25,208,245,173,200,192,136
60790 DATA 145,250, 32, 49,192,202,208
60800 DATA 225, 96, 32,182,192, 32,199
60810 DATA 193, 32,165,192, 76,199,193
60820 DATA 162, 21,160, 24,177,250,141
60830 DATA 200,192,136,177,250,200,145
60840 DATA 250,136,136,208,247,173,200
60850 DATA 192,200,145,250, 32, 49,192
60860 DATA 202,208,227, 96,  0,  0,  0
60870 DATA   0,128, 64, 32, 16,  8,  4
60880 DATA   2,  1, 32, 21,195, 32,182
60890 DATA 192,169,  0,141,231,193,141
60900 DATA 234,193,141,233,193, 24,169
60910 DATA   3,141,232,193,162,  0,160
60920 DATA   1,177,250,201, 32,240, 10
60930 DATA 189,235,193, 24,109,233,193
60940 DATA 141,233,193,200,232,224,  8
60950 DATA 208,234,152, 72,172,231,193
60960 DATA 173,233,193,145,248,153, 64
60970 DATA   3,238,231,193,104,168,169
60980 DATA   0,141,233,193,162,  0,206
60990 DATA 232,193,208,204, 32, 49,192
61000 DATA 238,234,193,173,234,193,201
61010 DATA  21,144,182,165,247,172,231
61020 DATA 193,145,248, 96, 32,182,192
61030 DATA 169,  0,160,160,133,252,132
61040 DATA 253, 32,115,194, 32,165,192
61050 DATA 169,  0,160,165,133,252,132
61060 DATA 253, 76,115,194,162, 21,160
61070 DATA  24,177,252,145,250,136,208
61080 DATA 249, 32, 49,192, 32,226,192
61090 DATA 202,208,238, 96,169,  0,160
61100 DATA 160,133,250,132,251,169,  0
61110 DATA 160,165,133,252,132,253,162
```

```
61120 DATA   4,160,  0,169, 32,145,250
61130 DATA 169,  0,145,252,200,208,245
61140 DATA 230,251,230,253,202,208,238
61150 DATA  96,165,  2,201,129,176,249
61160 DATA 120,169, 53,133,  1, 32,182
61170 DATA 192, 32, 21,195,169,  0,141
61180 DATA 231,193,141,234,193,169,  3
61190 DATA 141,232,193,160,  1,152, 72
61200 DATA 172,231,193,177,248,153, 64
61210 DATA   3,141,233,193,104,168,162
61220 DATA   0,189,235,193, 45,233,193
61230 DATA 240,  3,169,160, 44,169, 32
61240 DATA 145,250,200,232,224,  8,208
61250 DATA 235,238,231,193,206,232,193
61260 DATA 208,210, 32, 49,192, 32,226
61270 DATA 192,238,234,193,173,234,193
61280 DATA 201, 21,144,187,169, 55,133
61290 DATA   1, 88, 76, 21,192,198,  2
61300 DATA 138, 72,165,  2,162,  0,134
61310 DATA 248,134,249,133,248,162,  6
61320 DATA  24,  6,248, 38,249,202,208
61330 DATA 249,165,248, 24,105,  0,144
61340 DATA   2,230,249,133,248,165,249
61350 DATA  24,105,224,133,249,104,170
61360 DATA 230,  2, 96,230,  2,165,  2
61370 DATA  24,201,128,144,  4,169,  1
61380 DATA 133,  2, 32,175,194, 76,189
61390 DATA 195,198,  2,208,  4,169,127
61400 DATA 133,  2, 32,175,194, 76,189
61410 DATA 195,169,  7,141, 21,208,169
61420 DATA   5,141, 39,208,141, 40,208
61430 DATA 169,248,141,  0,208,141,  2
61440 DATA 208,169,131,141,  1,208,169
61450 DATA 177,141,  3,208,169, 13,141
61460 DATA 248,  7,141,249,  7,169, 11
61470 DATA 141,250,  7,169,  1,141, 41
61480 DATA 208,169,  2,141, 23,208,141
61490 DATA  29,208, 32, 46,196,169, 19
61500 DATA 169,  1,133,  2,169,  0,141
61510 DATA  32,208,141, 33,208, 32,156
```

```
61520 DATA 196, 32,208,196, 32,221,197
61530 DATA 169,128,141,138,  2,164,  2
61540 DATA  32,162,179, 32,221,189,160
61550 DATA  26,162,  3,134,214,132,211
61560 DATA  32,108,229,169,246,160,195
61570 DATA  32, 30,171,162,  1,189,  0
61580 DATA   1,201,  0,240, 11, 32,210
61590 DATA 255,232, 76,217,195,169,  1
61600 DATA 133, 89,169, 32, 32,210,255
61610 DATA 232,224,  5,208,248, 96, 83
61620 DATA  80, 82, 84, 46, 35, 58,  0
61630 DATA  32,228,255,240,251,162,  0
61640 DATA 221,172,198,240,  8,232,224
61650 DATA  27,208,246, 76,254,195,138
61660 DATA  10,170,189,197,198,133,250
61670 DATA 189,198,198,133,251,169,196
61680 DATA  72,169, 39, 72,108,250,  0
61690 DATA  32,243,193, 76,254,195,162
61700 DATA   0,138,157,192,  2,232,224
61710 DATA  64,208,248,169,255,162, 18
61720 DATA 157,192,  2,232,232,232,224
61730 DATA  24,144,246,169,  1,133, 87
61740 DATA 133, 88,133, 89,169,  5,133
61750 DATA 247, 32, 89,196, 76, 21,192
61760 DATA 165, 87,201,  1,176,  6,169
61770 DATA  24,133, 87,198, 88,201, 25
61780 DATA 144,  6,169,  1,133, 87,230
61790 DATA  88,165, 88,201,  1,176,  4
61800 DATA 169, 21,133, 88,201, 22,144
61810 DATA   4,169,  1,133, 88,165, 87
61820 DATA  10, 10, 10, 24,105, 24, 72
61830 DATA 165, 88, 10, 10, 10, 24,105
61840 DATA  66,141,  5,208,104,141,  4
61850 DATA 208, 76,111,197,165, 89,201
61860 DATA  16,144,  4,169,  0,133, 89
61870 DATA 168, 32,162,179, 32,221,189
61880 DATA 160, 26,162,  4,134,214,132
61890 DATA 211, 32,108,229,169,  6,160
61900 DATA 197, 32, 30,171,162,  1,189
61910 DATA   0,  1,240,  7, 32,210,255
```

```
61920 DATA 232, 76,193,196, 76,235,195
61930 DATA 165,247,201, 16,144,  4,169
61940 DATA   0,133,247,168, 32,162,179
61950 DATA  32,221,189,160, 26,162,  5
61960 DATA 134,214,132,211, 32,108,229
61970 DATA 169, 14,160,197, 32, 30,171
61980 DATA 162,  1,189,  0,  1,201,  0
61990 DATA 240,  7, 32,210,255,232, 76
62000 DATA 245,196, 76,235,195, 67, 45
62010 DATA  67, 79, 76, 32, 58,  0, 83
62020 DATA  45, 67, 79, 76, 32, 58,  0
62030 DATA 169, 32, 44,169,160, 72,165
62040 DATA  87, 24, 72,166, 88,202,138
62050 DATA  10,170,189, 66,197,133,250
62060 DATA 189, 67,197,133,251,104, 24
62070 DATA 101,250,133,250,144,  2,230
62080 DATA 251,160,  0,104,145,250, 76
62090 DATA   1,199,120,  4,160,  4,200
62100 DATA   4,240,  4, 24,  5, 64,  5
62110 DATA 104,  5,144,  5,184,  5,224
62120 DATA   5,  8,  6, 48,  6, 88,  6
62130 DATA 128,  6,168,  6,208,  6,248
62140 DATA   6, 32,  7, 72,  7,112,  7
62150 DATA 152,  7,192,  7,  0,164, 88
62160 DATA  32,162,179, 32,221,189,162
62170 DATA  26,160,  1,134,211,132,214
62180 DATA  32,108,229,162,  1,189,  0
62190 DATA   1,201,  0,240,  7, 32,210
62200 DATA 255,232, 76,132,197,224,  3
62210 DATA 240,  5,169, 32, 32,210,255
62220 DATA 164, 87, 32,162,179, 32,221
62230 DATA 189,162, 36,160,  1,134,211
62240 DATA 132,214, 32,108,229,162,  1
62250 DATA 189,  0,  1,201,  0,240,  7
62260 DATA  32,210,255,232, 76,176,197
62270 DATA 224,  3,240,  5,169, 32, 32
62280 DATA 210,255, 96,169, 25,133,211
62290 DATA  76,108,229,169, 18,133,211
62300 DATA  76,108,229,169, 27,133,211
```

```
62310 DATA  76,108,229,169,128,133,  2
62320 DATA  32, 61,192, 32,243,193, 32
62330 DATA 189,195,198,  2,208,246,169
62340 DATA   1,133,  2, 76,189,195, 32
62350 DATA  74,193,169, 54,133,  1, 32
62360 DATA   7,198,169, 55,133,  1, 32
62370 DATA 243,193, 96,169,120,160,  4
62380 DATA 133,250,132,251,169, 32,160
62390 DATA 163,133,252,132,253,162, 21
62400 DATA 160, 24,177,252,145,250,136
62410 DATA 208,249, 32, 49,192,165,252
62420 DATA  56,233, 40,133,252,176,  2
62430 DATA 198,253, 24,202,208,229, 96
62440 DATA  32, 74,193,169, 54,133,  1
62450 DATA  32, 70,198,169, 55,133,  1
62460 DATA  76,243,193,169,  0,133,252
62470 DATA 169,160,133,253,162, 21,169
62480 DATA 152,160,  7,133,250,132,251
62490 DATA 160, 21,177,252,141,154,198
62500 DATA 152, 72,138,168,173,154,198
62510 DATA 145,250,104,168,165,250, 56
62520 DATA 233, 40,133,250,176,  2,198
62530 DATA 251,136,208,226, 32,226,192
62540 DATA 202,208,210,162, 21,169,120
62550 DATA 160,  4,133,250,132,251,160
62560 DATA  24,169, 32,145,250,136,192
62570 DATA  21,208,247, 32, 49,192,202
62580 DATA 208,239, 96,  0, 32, 21,195
62590 DATA 160, 64,185, 64,  3,145,248
62600 DATA 136,208,248, 76,175,194,  0
62610 DATA  17,145, 29,157,147, 19, 43
62620 DATA  45, 89, 66, 71, 72, 83, 82
62630 DATA  24,141, 32,160, 73, 70,133
62640 DATA 137,134,138, 13,247,198,252
62650 DATA 198,  1,199,  6,199, 83,199
62660 DATA  11,199, 68,195, 87,195,104
62670 DATA 192,238,192,141,193,187,193
62680 DATA 245,197, 53,198, 20,199, 20
62690 DATA 199, 25,197, 22,197, 74,192
```

```
62700 DATA  33,199, 46,199, 57,199, 62
62710 DATA 199, 78,199, 89,199,230, 88
62720 DATA  76, 89,196,198, 88, 76, 89
62730 DATA 196,230, 87, 76, 89,196,198
62740 DATA  87, 76, 89,196,169,  1,133
62750 DATA  87,133, 88, 76, 89,196,104
62760 DATA 104,169,  0,141, 21,208,169
62770 DATA 147, 32,210,255, 96,169,120
62780 DATA 160,  4,133,250,132,251,169
62790 DATA 160, 76, 31,192,230, 89, 32
62800 DATA 156,196,165, 89,141, 41,208
62810 DATA  96,198, 89, 76, 48,199,230
62820 DATA 247, 32,208,196,165,247,141
62830 DATA  39,208,141, 40,208, 76, 21
62840 DATA 192,198,247, 76, 64,199, 32
62850 DATA  61,192, 76, 11,199,230, 88
62860 DATA 169,  1,133, 87, 76, 89,196
62870 DATA 169,  7,141, 21,208, 32,175
62880 DATA 194, 32, 89,196, 32,156,196
62890 DATA  32,208,196, 76,189,195,120
62900 DATA 169, 48,133,  1,169,  0,160
62910 DATA   4,133,250,132,251,169,  0
62920 DATA 160,208,133,252,132,253,160
62930 DATA   0,162,  3, 96, 32,118,199
62940 DATA 177,250,145,252,200,208,249
62950 DATA 230,251,230,253,202, 16,242
62960 DATA 169, 55,133,  1, 88, 96, 32
62970 DATA 118,199,177,252,145,250,200
62980 DATA 208,249,230,251,230,253,202
62990 DATA  16,242, 76,161,199,169,  0
63000 DATA 160,216,133,250,132,251,162
63010 DATA   3,173,134,  2,160,  0,145
63020 DATA 250,200,208,251,230,251,202
63030 DATA  16,246, 96,  0,111,  0,111
```

2.6.3 Erklärung des Spriteeditors

Kommen wir nur zur Anleitung des Spritegenerators. Nach dem Start desselben, werden Sie zunächst ein kurzes Aufbauen des Editorfeldes bemerken, das aber sofort wieder verschwindet. Dies hängt mit der internen Verwaltung der Bildschirmverwaltung zusammen. Anschließend ergibt sich folgender Bildschirmaufbau:

```
          Spritegenerator



     Sprite editieren.....1

     Sprite laden.....2

     Sprite speichern.....3
```

Zunächst die Funktionen des Spriteeditors, die durch Tastendruck angewählt werden können.

Taste	Funktion
"crsr"	Mit diesen Tasten können Sie sich beliebig im Spriteeditorfeld umherbewegen. Die Position des Cursors wird ständig in der linken, oberen Ecke des Editorfeldes angezeigt.
"home"	Durch einen Druck auf diese Taste positionieren Sie den Cursor in der linken oberen Ecke.

"clr"	Mit der "CLR"-Taste können Sie Ihre Spritedefinition löschen. Gleichzeitig wird der Cursor in der linken, oberen Ecke positioniert.
"RETURN"	Durch die RETURN-Taste springt der Cursor an den Anfang der nächsten Zeile.
"y/g/b/h"	Durch diese Tasten können Sie Ihr Sprite in beliebige Richtungen scrollen. Gescrollt wird dabei jeweils in die Richtung, in der der Buchstabe auf der Tastatur weist.
"s"	Mit dieser Taste können Sie Ihr Sprite an der X-Achse spiegeln.
"r"	Hierduch kann das Sprite im Uhrzeigersinn gedreht werden. Dabei muß jedoch beachtet werden, daß die letzten 3 Spalten, also die Spalten 21-24 Ihres Sprites verloren gehen.
CTRL/"x"	Mit dieser Tastenkombination können Sie den Spriteeditor verlassen. Als Ersatz dafür können Sie auch die Tastenkombination SHIFT & RETURN benutzen.

Allgemeines zum Spriteeditor:

Grundsätzlich werden alle Operationen die Sie ausführen, sofort auch auf die Sprites übertragen, die sich in Orginalgrößen neben dem eigentlichen Editorfeld befinden.

Insgesamt können Sie 127 verschiedene Sprites auf einmal im Speicher halten. Umschalten zwischen den einzelnen Definitionen können Sie mit der "+" und der "-" Taste. So wird es auch sehr einfach, Sprites für Animationen zu entwerfen.

I/O-Operationen:

Mit den Menuepunkten 2 und 3 können Sie Ihr Sprite von Diskette laden oder auf Diskette abspeichern. Grundsätzlich wird die Spritenummer geladen oder gespeichert, die Sie zuletzt editiert haben.

2.7 Rasterzeileninterrupt - Der geteilte Bildschirm

Eine Eigenschaft des Videocontrollers im Commodore 64, die sehr schöne Effekte ermöglicht, ist noch wenig beschrieben und benutzt worden - der Rasterinterrupt.

Um Ihnen zu erklären, was es damit auf sich hat, müssen wir etwas weiter ausholen und uns klar machen, wie der Videocontroller Ihres Rechners ein Fernsehbild erzeugt.

Das Fernsehbild wird dazu aus einzelnen Zeilen aufgebaut, die Sie deutlich erkennen können, wenn Sie das Fernsehbild einmal näher betrachten. Sie können dann auch erkennen, daß z.B. ein Buchstabe aus acht Zeilen aufgebaut ist. Der Videocontroller hat nun ein Register, das immer die Nummer der Fernsehzeile enthält, die gerade dargestellt wird. Dieses Register hat die Nummer 18, ist also unter der Adresse 53248+18 = 53266 zu erreichen. Fragen Sie den Inhalt dieses Register ab,

```
PRINT PEEK(53266)
```

so erhalten Sie zufällige Werte, die die augenblicklich dargestellte Rasterzeile angeben. Da in einer Sekunde jedoch 25 Fernsehbilder dargestellt werden, können wir dies in BASIC nicht schnell genug auswerten. Wir müssen also zur Program-

mierung in Maschinensprache übergehen. Dabei kommt uns noch eine Eigenschaft des Videocontrollers zunutze: Der Videocontroller ist in der Lage, ein laufendes Programm dann zu unterbrechen, wenn gerade eine bestimmte Rasterzeile dargestellt wird. Die Nummer dieser Rasterzeile können Sie durch Schreiben in das gleiche Register des Videocontrollers bestimmen. Eine solche automatische Programmunterbrechung, wir haben dies ja in dem Kapitel über den Chargenerator kennengelernt, nennt man 'Interrupt'. Weiterhin muß der Interrupt durch den Vidoecontroller zugelassen werden. Dies geschieht durch beschreiben des Interruptmaskenregisters. Eine solche Programmierung ist jedoch nur in Maschinensprache möglich. Wir haben deshalb ein kleines Programm für Sie geschrieben, das diese Möglichkeiten ausnutzt.

```
10:   033C                    .OPT P1
20:   033C          IRQALT    =    $EA31
25:   033C          IRQVECT   =    $314
35:   033C          RASTER    =    $D012    ; Rasterzeile
40:   033C          IRQREG    =    $D019    ; Flag für Videointerrupt
50:   033C          MASKE     =    $D01A    ; Interruptmaske des
                                              ; Videocontrollers
55:   033C          RAHMEN    =    $D020    ; Rahmenfarbe
60:   033C          FARBE     =    $D021    ; Hintergrundfarbe
100:  033C          ICR       =    $DC0D    ; Flag für Timerinterrupt
110:  033C          RETIRQ    =    $FEBC    ; Rückkehr vom Interrupt
120:  00FB                    *=   $FB
130:  00FC          ZEILE1    *=   *+1
140:  00FD          ZEILE2    *=   *+1
150:  00FE          FARBE1    *=   *+1
160:  00FF          FARBE2    *=   *+1
;
190:  033C                    *=   828      ; Kassettenpuffer
200:  033C 78       SETUP     SEI
210:  033D A9 5B              LDA  #< IRQNEU
220:  033F 8D 14 03           STA  IRQVECT
230:  0342 A9 03              LDA  #> IRQNEU
240:  0344 8D 15 03           STA  IRQVECT+1
260:  0347 A5 FB              LDA  ZEILE1
```

```
270:  0349 8D 12 D0          STA  RASTER  ; Rasterzeile für Interrupt
275:  034C AD 11 D0          LDA  RASTER-1
275:  034F 29 7F             AND  #$7F    ; Highbyte löschen
275:  0351 8D 11 D0          STA  RASTER-1
280:  0354 A9 81             LDA  #$81    ; IRQ durch Raster erlauben
300:  0356 8D 1A D0          STA  MASKE
310:  0359 58                CLI
320:  035A 60                RTS
;
400:  035B AD 19 D0 IRQNEU   LDA  IRQREG
400:  035E 8D 19 D0          STA  IRQREG  ; Interruptflag löschen
410:  0361 29 01             AND  #1
420:  0363 D0 07             BNE  SCREEN  ; Interrupt durch
                                          ; Rasterzeile ?
430:  0365 AD 0D DC          LDA  ICR     ; Interruptflag löschen
430:  0368 58                CLI          ; Rasterinterrupt zulassen
435:  0369 4C 31 EA          JMP  IRQALT
;
450:  036C AD 12 D0 SCREEN   LDA  RASTER  ; Rasterzeile lesen
460:  036F C5 FC             CMP  ZEILE2
470:  0371 B0 0D             BCS  ZWEITER ; größer oder gleich 2.
                                          ; Wert ?
480:  0373 A5 FD             LDA  FARBE1
490:  0375 8D 20 D0          STA  RAHMEN  ; Farbe 1 setzen
500:  0378 8D 21 D0          STA  FARBE
510:  037B A5 FC             LDA  ZEILE2  ; nächsten Interrupt bei 2.
                                          ; Wert
520:  037D 4C 8A 03          JMP  EXIT
560:  0380 A5 FE    ZWEITER  LDA  FARBE2
570:  0382 8D 20 D0          STA  RAHMEN  ; Farbe 2 setzen
580:  0385 8D 21 D0          STA  FARBE
590:  0388 A5 FB             LDA  ZEILE1  ; nächsten Interrupt
600:  038A 8D 12 D0 EXIT     STA  RASTER  ; bei Zeile 1
610:  038D 4C BC FE          JMP  RETIRQ
```

Hier wieder ein Ladeprogramm in BASIC. Geben Sie dieses Programm zuerst ein, speichern Sie es ab und starten Sie es mit 'RUN'.

```
100 FOR I = 828 TO 911
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 120,169, 91,141, 20,  3,169,  3,141, 21,  3,165
130 DATA 251,141, 18,208,173, 17,208, 41,127,141, 17,208
140 DATA 169,129,141, 26,208, 88, 96,173, 25,208,141, 25
150 DATA 208, 41,  1,208,  7,173, 13,220, 88, 76, 49,234
160 DATA 173, 18,208,197,251,176, 13,165,253,141, 32,208
170 DATA 141, 33,208,165,252, 76,138,  3,165,254,141, 32
180 DATA 208,141, 33,208,165,251,141, 18,208, 76,188,254
190 IF S <> 10677 THEN PRINT "FEHLER IN DATAS !!" : END
200 PRINT "OK"
```

Als Beispiel hierzu haben wir ein Programm für Sie entwickelt, das es Ihnen ermöglicht, einen Bildschirmbereich mit einer anderen Hintergrundfarbe zu versehen. Sie können dadurch eine oder mehrere Bildschirmzeilen hervorheben. Um das Programm möglichst universell zu halten, können Sie sowohl die Hintergrundfarben des hervorzuhebenden Bereichs als auch die des übrigen Bereichs durch POKE-Befehle bestimmen. Ebenso können Sie die Rasterzeile, bei der die Umschaltung auf die zweite Hintergrundfarbe geschehen soll bestimmen. Dies gilt natürlich auch für die Nummer der Rasterzeile, bei der wieder auf die erste Farbe zurückgeschaltet wird. Dies mag zwar etwas kompliziert klingen; probieren Sie jedoch das nachfolgende Programm einmal aus und Sie werden den Effekt sehen.

Mit diesem Programm können Sie einen farbigen Balken von der Breite einer Bildschirmzeile (8 Rasterzeilen) durch Drücken der Tasten 'Cursor hoch' und 'Cursor runter' um jeweils eine Zeile nach oben oder unten bewegen. Beachten Sie, daß dies nicht auf

einmal geschieht, sondern um 8 Rasterzeilen nacheinander. Die Funktionstasten können Sie benutzen, um die Farbe des Balkens und des übrigen Hintergrunds zu ändern.

```
100 Z1=251:Z2=Z1+1:F1=Z2+1:F2=F1+1
110 Z=50:SYS828:REM INTERRUPT INITIALISIEREN
120 POKEZ1,Z:POKEZ2,Z+8:POKEF1,6:POKEF2,8
150 GETA$:IFA$=""THEN150
160 IFA$=CHR$(17)THENGOSUB200
170 IFA$=CHR$(145)THENGOSUB300
180 IFA$=CHR$(133)THENGOSUB400
190 IFA$=CHR$(134)THENGOSUB500
195 GOTO150
200 IFZ<240THENFORI=0TO7:Z=Z+1:POKEZ1,Z:POKEZ2,Z+8:NEXT
210 RETURN
300 IFZ>50THENFORI=0TO7:Z=Z-1:POKEZ1,Z:POKEZ2,Z+8:NEXT
310 RETURN
400 POKEF1,PEEK(F1)+1AND15:RETURN
500 POKEF2,PEEK(F2)+1AND15:RETURN
```

Sie können das Programm für Ihre Zwecke abändern, indem Sie in die Speicherstellen 251 bis 254 der Reihe nach die Nummer der Rasterzeile poken, bei der die Umschaltung auf die zweite Farbe stattfinden soll, dann die Rasterzeile, bei der wieder auf die ursprüngliche Farbe zurückgeschaltet wird. Die nächsten beiden Adressen 253 und 254 enthalten den Farbkode der ersten und zweiten Farbe. Für die Rasterzeilen gilt folgender Bereich:

Die Rasterzeile 50 entspricht der oberen Bildbegrenzung (dort wo der Rahmen beginnt), während der Beginn des unteren Rahmens der Rasterzeile 250 entspricht. Eine Bildschirmzeile ist in 8 Rasterzeilen aufgeteilt. Dadurch können Sie auch die Grenze zwischen zwei verschiedenen Farben mitten in eine Bildschirmzeile legen.

Die Umschaltung der Hintergrundfarbe ist jedoch nicht der einzige Effekt, den Sie mit dem Rasterinterrupt erreichen können. Alle Parameter des Videocontrollers können Sie interruptgesteuert verändern. Hier ein paar Anregungen:

Blenden Sie in eine hochauflösende Grafik eine Textzeile ein oder schalten Sie zwischen zwei Grafikseiten um. Sie sehen dann z.B. in der oberen Hälfte des Bildschirms ein Grafikbild und in der unteren Hälfte ein zweites. Durch stetiges Ändern der Rasterzeile, bei der umgeschaltet wird, können Sie so z.B. zwei Grafikbilder von oben nach unten überblenden, ähnlich wie in unserem Beispiel der farbige Balken. Ihrer Phantasie sind dabei keine Grenzen gesetzt; vielleicht probieren Sie auch einmal verschiedene Zeichensätze in verschiedenen Teilen des Bildschirms!

Mit dieser Technik können Sie auch Effekte erreichen, die der Commodore 64 von Hause aus eigentlich gar nicht kann, z.B. ist es mittels des Rasterinterrupts möglich, mehr als 8 Sprites gleichzeitig darzustellen. Dazu können Sie in der oberen Hälfte des Bildschirms acht Sprites darstellen. Beim Erreichen einer bestimmten Rasterzeile setzen Sie einfach die Spritepointer und -Koordinaten um und in der unteren Hälfte erscheinen weitere 8 Sprites. Natürlich können Sie den Bildschirm dazu auch in mehr als zwei Teile aufteilen.

Um Ihnen das Experimentieren zu erleichtern haben wir das folgende Programm für Sie entwickelt. Das Programm teilt den Bildschirm in 5 verschiedene Bildschirmzonen ein, die Sie unabhängig voneinander mit allen Möglichkeiten des Videocontrollers manipulieren können.

Dazu wird im RAM des Computers 5mal eine Kopie der eigentlichen Inhalte der Videocontrollerregister angelegt, die dann, wenn die von Ihnen festgelegten Rasterzeilen gerade aufgebaut werden, in den VIC kopiert werden.

Die Bildschirmzonen selbst können Sie durch die Anweisung

```
SYS 49152,Kopie,Rasterzeile
```

festlegen.

Ihnen stehen hierbei die Kopien 1 bis 5 zur Verfügung. Jeder andere Wert außer Null, der zum Ausschalten des Extended-VIC Programms dient, führt zu einem Fehler. Die Rasterzeilen sind im Bereich von 0 bis 511 zu wählen, wobei darauf geachtet werden sollte, daß die jeweils höher gelegene VIC-Kopie auch bei der höheren Rasterzeile eingelesen wird, also beispielsweise so:

```
SYS 49152,1,50
SYS 49152,2,100
SYS 49152,3,150
u.s.w.
```

Dagegen führt folgendes zu einem enervierenden Bildschirmflimmern:

```
SYS 49152,1,150
SYS 49152,2,50
```

Achten Sie also auf die Werte die Sie wählen. Vor allen I/O Operationen sollten Sie das Programm durch RUN/STOP RESTORE oder SYS 49152,0 desaktivieren, da es sonst zu Timingproblemen kommen kann.

Hier nun zuerst das dokumentierte Assemblerlisting und dann der BASIC-Lader.

```
110:  c000                    .opt p1,oo
120:                  ;
130:                  ;*****************************
140:                  ;*   extended-vic programm   *
150:                  ;*****************************
160:                  ;
170:  c000                    *=   $c000
180:                  ;
190:  c000 4c 06 c0           jmp  start   ; zum programmstart
200:                  ;
210:  b79e            getbyt  =    $b79e   ; byte holen
220:  ad8a            frmnum  =    $ad8a   ; formel auswerten
230:  b7f7            getadr  =    $b7f7   ; 16-bit wert holen
240:  aefd            chkcom  =    $aefd   ; auf komma testen
250:  0079            chrgot  =    $0079   ; letztes zeichen
260:  0073            chrget  =    $0073   ; naechstes zeichen
270:                  ;
280:  d000            v       =    53248   ; basisadresse vic
290:                  ;
300:  c003 4c 48 b2 error     jmp  $b248   ; illegal quantity
310:                  ;
320:  c006 20 fd ae start     jsr  chkcom  ; komma holen
330:  c009 20 9e b7           jsr  getbyt  ; byte wert holen
340:  c00c e0 06              cpx  #6      ; testen ob erlaubt
350:  c00e b0 f3              bcs  error   ; zahl zu hoch
360:  c010 e0 00              cpx  #0      ; null
370:  c012 f0 11              beq  off     ; bei null irq aus
380:  c014 8a                 txa
380:  c015 48                 pha
380:  c016 ad 14 03           lda  $0314
390:  c019 c9 b5              cmp  #<neuirq  ; schon raster aktiv
400:  c01b f0 03              beq  ok        ; ja
410:  c01d 20 37 c0           jsr  init      ; raster initialisieren
420:  c020 68         ok      pla
420:  c021 aa                 tax
420:  c022 4c 80 c0           jmp  setirq
```

```
430:                ;
440:                ;***********
450:                ;*   off   *
460:                ;***********
470:                ;
480:  c025 78       off     sei          ; interrupt verhindern
490:  c026 a9 31            lda  #<$ea31 ; low-byte irq
500:  c028 a0 ea            ldy  #>$ea31 ; high-byte irq
510:  c02a 8d 14 03         sta  $0314   ; interruptvektor low
520:  c02d 8c 15 03         sty  $0315   ; interruptvektor high
530:  c030 a9 80            lda  #%10000000  ; maske fuer imr
540:  c032 8d 1a d0         sta  v+26    ; und loeschen
550:  c035 58               cli          ; interrupt freigeben
560:  c036 60               rts          ; wieder zum basic
570:                ;
580:                ;*********************************
590:                ;* initialisieren des raster-irq *
600:                ;*********************************
610:                ;
620:  c037 78       init    sei           ; interrupt verhindern
630:  c038 a9 b5            lda  #<neuirq ; low-byte neuirq
640:  c03a a0 c0            ldy  #>neuirq ; high byte neuirq
650:  c03c 8d 14 03         sta  $0314    ; in irq-vektor low
660:  c03f 8c 15 03         sty  $0315    ; in irq-vektor high
670:  c042 a0 00            ldy  #0       ; yr-index auf null
680:  c044 b9 00 d0 loop0   lda  v,y      ; orginalen vic
690:  c047 c0 11            cpy  #17      ; register 17
700:  c049 d0 04            bne  ok1      ; nein!
710:  c04b 29 7f            and  #%01111111  ; bit 7 loeschen
720:  c04d d0 0c            bne  ok9      ; nach ok2
730:  c04f c0 12    ok1     cpy  #18      ; register 18
740:  c051 d0 02            bne  ok2      ; nein!
750:  c053 a9 00            lda  #0
760:  c055 c0 1a    ok2     cpy  #26
760:  c057 d0 02            bne  ok9
760:  c059 a9 81            lda  #%10000001
770:  c05b 99 f7 c0 ok9     sta  vic1,y
780:  c05e 99 28 c1         sta  vic2,y  ; in kopie iii
790:  c061 99 59 c1         sta  vic3,y  ; in kopie ii
800:  c064 99 8a c1         sta  vic4,y  ; in kopie  iv
```

```
810:  c067 99 bb c1          sta  vic5,y  ; in kopie   v
820:  c06a c8                iny          ; index erhoehen
830:  c06b c0 31             cpy  #49     ; schon alle register
840:  c06d d0 d5             bne  loop0   ; nein!
850:                ;
860:  c06f a9 00             lda  #0      ; vic kopie 0
870:  c071 85 02             sta  2       ; index-zeiger
880:  c073 a9 81             lda  #%10000001  ; raster irq
890:  c075 8d 1a d0          sta  v+26    ; ins imr schreiben
900:  c078 ad 19 d0          lda  v+25    ; irr auslesen
910:  c07b 8d 19 d0          sta  v+25    ; und loeschen
920:  c07e 58                cli          ; interrupt freigeben
930:  c07f 60                rts          ; und wieder zurueck
940:                ;
950:                ;********************************
960:                ;*   zeile fuer irq festlegen   *
970:                ;********************************
980:                ;
990:  c080 8a       setirq   txa          ; und nach akku bringen
1000: c081 0a                asl          ; verdoppeln und
1010: c082 aa                tax          ; wieder nach x bringen
1020: c083 bd eb c0          lda  tab1,x  ; low-byte vic-kopie
1030: c086 85 f7             sta  $f7
1040: c088 bd ec c0          lda  tab1+1,x  ; high-byte vic-kopie
1050: c08b 85 f8             sta  $f8
1060:               ;
1070: c08d 20 fd ae          jsr  chkcom  ; auf komma testen
1080: c090 20 8a ad          jsr  frmnum  ; formelauswertung
1090: c093 20 f7 b7          jsr  getadr  ; 16-bit wert holen
1100: c096 a5 15             lda  $15     ; highbyte 16-bitwert
1110: c098 c9 02             cmp  #2      ; groesser gleich 2
1120: c09a 90 03             bcc  ok4     ; in ordnung
1130: c09c 4c 03 c0          jmp  error   ; illegal quantity
1140: c09f 78       ok4      sei          ; interrupt sperren
1150: c0a0 4a                lsr  a       ; bit 0 ins carry
1160: c0a1 6a                ror  a       ; carry ins 7. bit
1170: c0a2 85 f9             sta  $f9     ; in adr. $f9 speichern
1180: c0a4 a0 11             ldy  #17     ; register 17
1190: c0a6 b1 f7             lda  ($f7),y ; register 17 laden
1200: c0a8 29 7f             and  #%01111111
```

```
1210:  c0aa 05 f9            ora  $f9     ; oder verknuepfen
1220:  c0ac 91 f7            sta  ($f7),y
1230:  c0ae a5 14            lda  $14     ; low-byte
1240:  c0b0 c8               iny          ; register 18
1250:  c0b1 91 f7            sta  ($f7),y
1260:  c0b3 58               cli          ; interrupt freigeben
1270:  c0b4 60               rts          ; wieder zum basic
1280:                ;
1290:                ;************************
1300:                ;*   neue irq-routine   *
1310:                ;************************
1320:                ;
1330:  c0b5 ad 19 d0 neuirq  lda  v+25
1330:  c0b8 8d 19 d0         sta  v+25
1340:  c0bb 30 07            bmi  raster
1340:  c0bd ad 0d dc         lda  $dc0d
1340:  c0c0 58               cli
1340:  c0c1 4c 31 ea         jmp  $ea31
1350:  c0c4 e6 02    raster  inc  2       ; index erhoehen
1360:  c0c6 a5 02            lda  2       ; einladen
1370:  c0c8 c9 05            cmp  #5      ; schon sechs
1380:  c0ca d0 04            bne  ok3     ; nein!
1390:  c0cc a9 00            lda  #0      ; index auf null setzen
1400:  c0ce 85 02            sta  2       ; und abspeichern
1410:  c0d0 0a       ok3     asl          ; verdoppeln
1420:  c0d1 a8               tay          ; akku nach yr
1430:  c0d2 b9 eb c0         lda  tab1,y  ; low-byte kopie
1440:  c0d5 85 fa            sta  $fa
1450:  c0d7 b9 ec c0         lda  tab1+1,y
1460:  c0da 85 fb            sta  $fb
1470:  c0dc a0 00            ldy  #0      ; index loeschen
1480:  c0de b1 fa    loop3   lda  ($fa),y ; kopie in
1490:  c0e0 99 00 d0         sta  v,y     ; vic kopieren
1500:  c0e3 c8               iny          ; index erhoehen
1510:  c0e4 c0 31            cpy  #49     ; alle register
1520:  c0e6 d0 f6            bne  loop3   ; nein
1530:  c0e8 4c bc fe         jmp  $febc
1540:                ;
1550:  c0eb bb c1 f7 tab1    .wordvic5,vic1,vic2,vic3,vic4,vic5
1560:                ;
```

```
1570:  c0f7          vic1     =    *
1580:  c128          vic2     =    vic1+49
1590:  c159          vic3     =    vic2+49
1600:  c18a          vic4     =    vic3+49
1610:  c1bb          vic5     =    vic4+49
1620:                ;
```

BASIC-Lader:

```
100 rem ****************************
110 rem *   extended-vic-program   *
120 rem ****************************
130 :
140 for i=49152 to 49398
150 :   read a
160 :   s=s+a
170 :   poke i,a
180 next i
190 :
200 if s<>32344 then print "fehler in data's!!!" : end
210 :
220 print "programm ok!!!"
230 :
240 end
250 :
260 data  76,  6,192, 76, 72,178, 32
270 data 253,174, 32,158,183,224,  6
280 data 176,243,224,  0,240, 17,138
290 data  72,173, 20,  3,201,181,240
300 data   3, 32, 55,192,104,170, 76
310 data 128,192,120,169, 49,160,234
320 data 141, 20,  3,140, 21,  3,169
330 data 128,141, 26,208, 88, 96,120
340 data 169,181,160,192,141, 20,  3
350 data 140, 21,  3,160,  0,185,  0
360 data 208,192, 17,208,  4, 41,127
370 data 208, 12,192, 18,208,  2,169
380 data   0,192, 26,208,  2,169,129
```

```
390 data 153,247,192,153, 40,193,153
400 data  89,193,153,138,193,153,187
410 data 193,200,192, 49,208,213,169
420 data   0,133,  2,169,129,141, 26
430 data 208,173, 25,208,141, 25,208
440 data  88, 96,138, 10,170,189,235
450 data 192,133,247,189,236,192,133
460 data 248, 32,253,174, 32,138,173
470 data  32,247,183,165, 21,201,  2
480 data 144,  3, 76,  3,192,120, 74
490 data 106,133,249,160, 17,177,247
500 data  41,127,  5,249,145,247,165
510 data  20,200,145,247, 88, 96,173
520 data  25,208,141, 25,208, 48,  7
530 data 173, 13,220, 88, 76, 49,234
540 data 230,  2,165,  2,201,  5,208
550 data   4,169,  0,133,  2, 10,168
560 data 185,235,192,133,250,185,236
570 data 192,133,251,160,  0,177,250
580 data 153,  0,208,200,192, 49,208
590 data 246, 76,188,254,187,193,247
600 data 192, 40,193, 89,193,138,193
610 data 187,193
```

Nachdem wir nun schon ein sehr komfortables Programm zum Arbeiten mit dem VIC-Interrupt haben, wollen wir dieses Programm auch entsprechend nutzen. Splitten Sie den Bildschirm bitte mit folgenden fünf Kommandos in fünf, etwa gleich große Rasterzonen:

```
SYS 49152,1,40
SYS 49152,2,90
SYS 49152,3,140
SYS 49152,4,190
SYS 49152,5,240
```

Dann definieren Sie bitte die Anfangsadressen der einzelnen VIC-Kopien wie folgt (Direktmodus):

```
V1=49399:V2=V1+49:V3=V2+49:V4=V3+49:V5=V4+49
```

Nun können Sie die einzelnen "Pseudo-VICs" wie gewohnt verändern, nur daß Sie jeweils eine andere Basisadresse benutzen müssen. Für die erste VIC-Kopie V1, für die zweite Kopie V2 u.s.w.

Teilen wir unseren Bildschirm doch einmal in eine Text, eine HRG und eine MC Screen. Dies könnte beispielsweise so aussehen (Basisadresse in den Variablen V1-V5):

```
POKE V2+17,PEEK (V1+17) OR 2^5 : rem grafik aktivieren
POKE V2+24,24 : rem grafikspeicher bei 8192 ($2000)
POKE V3+17,PEEK (V2+17) OR 2^5 : rem grafik aktivieren
POKE V3+24,24 : rem grafikspeicher bei 8192 ($2000)
POKE V3+22,PEEK (V2+22) OR 2^4 : rem multi color modus
```

Und schon stellen die oberen drei Bereiche HI-RES und die unteren beiden Bereiche Text dar. Um Ihnen zu demonstrieren, wie die Verwaltung von 32 Sprites aussehen kann, hier noch eine Spritedemo:

Spritedemo:

```
100 rem ****************************
110 rem *   extended-vic-program   *
120 rem *   --------demo--------   *
130 rem ****************************
140 :
150 print chr$(147):
160 :
```

```
170 v(1) =   49399 : rem adr. kopie (1)
180 v(2) = v(1)+49 : rem adr. kopie (2)
190 v(3) = v(2)+49 : rem adr. kopie (3)
200 v(4) = v(3)+49 : rem adr. kopie (4)
210 v(5) = v(4)+49 : rem adr. kopie (5)
220 :
230 sys 49152,1, 41 : rem irq-zeile (1)
240 sys 49152,2, 91 : rem irq-zeile (2)
250 sys 49152,3,141 : rem irq-zeile (3)
260 sys 49152,4,191 : rem irq-zeile (4)
270 sys 49152,5,240 : rem irq-zeile (5)
280 :
290 for i=1 to 5
300 :   for j=0 to 48
310 :       read a
320 :       if j=17 then a=a and 127
330 :       if j=18 then 350
340 :       poke v(i)+j,a
350 :   next j
360 next i
370 :
380 for i=0 to 62
390 :   read a
400 :   poke 704+i,a
410 next i
420 :
430 for i=2040 to 2047 : poke i,11 : next i
440 t=peek(v(5)+33)
450 for i=4 to 1 step -1
460 :   poke v(i+1)+33,peek(v(i)+33)
470 :   poke v(i+1)+32,peek(v(i)+32)
480 next i
490 :
500 poke v(1)+33,t
510 poke v(1)+32,t
520 :
530 for pa=0 to 250 : next pa
540 :
550 goto 440
560 :
```

```
570 rem ******************
580 rem *   vic data's   *
590 rem ******************
600 :
610 data   0,  0,  0,  0
620 data   0,  0,  0,  0
630 data   0,  0,  0,  0
640 data   0,  0,  0,  0
650 data   0, 27, 41,209
660 data   0,255,200,  0
670 data  21,121,129,255
680 data 255,  0,  0,255
690 data   0,  0,241,242
700 data 243,  2,  7,  6
710 data   6,  6,  6,  6
720 data   6,  6,  6,255
730 data 253
740 rem ---------------
750 data  50, 60, 79, 60
760 data 108, 60,137, 60
770 data 167, 60,196, 60
780 data 225, 60,254, 60
790 data   0, 27, 91,209
800 data   0,255,200,  0
810 data  21,121,129,255
820 data 255,  0,  0,255
830 data   2,  2,241,242
840 data 243,  2,  7,  6
850 data   6,  6,  6,  6
860 data   6,  6,  6,255
870 data 252
880 rem ---------------
890 data  50,110, 79,110
900 data 108,110,137,110
910 data 167,110,196,110
920 data 225,110,254,110
930 data   0, 27,191,209
940 data   0,255,200,  0
950 data  21,121,129,255
960 data 255,  0,  0,255
```

```
970 data   7,  7,241,242
980 data 243,  2,  7,  6
990 data   6,  6,  6,  6
1000 data  6,  6,  6,255
1010 data 252
1020 rem ---------------
1030 data  50,160, 79,160
1040 data 108,160,137,160
1050 data 167,160,196,160
1060 data 225,160,254,160
1070 data  0, 27,240,209
1080 data  0,255,200,  0
1090 data  21,121,129,255
1100 data 255,  0,  0,255
1110 data  4,  4,241,242
1120 data 243,  2,  7,  6
1130 data  6,  6,  6,  6
1140 data  6,  6,  6,255
1150 data 252
1160 rem ---------------
1170 data  50,210, 79,210
1180 data 108,210,137,210
1190 data 167,210,196,210
1200 data 225,210,254,210
1210 data  0, 27,240,209
1220 data  0,255,200,  0
1230 data  21,121,129,255
1240 data 255,  0,  0,255
1250 data  5,  5,241,242
1260 data 243,  2,  7,  6
1270 data  6,  6,  6,  6
1280 data  6,  6,  6,255
1290 data 252
1300 :
1310 rem **********************
1320 rem *   sprite-data's   *
1330 rem **********************
1340 :
```

```
1350 data   0,  0,  0
1360 data   0,  0,  0
1370 data   0,  0,  0
1380 data   0,  0,  0
1390 data   0,  0,  0
1400 data   0,  0,  0
1410 data   0,  0,  0
1420 data   0,  0,  0
1430 data   0,  0,  0
1440 data   0,  0,  0
1450 data   0,  0,  2
1460 data   0,  0, 10
1470 data   0,  0, 42
1480 data   0,192,170
1490 data   3, 50,170
1500 data  10,170,160
1510 data  42,170,165
1520 data 170,170,175
1530 data 171,255,229
1540 data  42,170,160
1550 data  10,170,170
```

3. Kapitel

Soundprogrammierung

3.1 Soundanimation

Alle hier vorgestellten Routinen und Tricks sind so ausgewählt, daß Sie diese direkt für Animationseffekte einsetzen können. Das bedeutet, daß auf bereits bekannte Tricks und Erweiterungen verzichtet wurde.

Derartige Grundlagen können Sie gesondert in einem der vielen speziellen Sound- und Musikbüchern zum Commodore nachlesen.

3.1.1 Datasette-Sound

Mit nur einer einzigen geringfügigen Modifikation können Sie die Datasette wie einen herkömmlichen Audio-Recorder betreiben. Sie können dann also alle auf der Cassette gespeicherten Signale auch über den Monitor-/TV-Lautsprecher ausgeben.

Um dieses zu realisieren, muß lediglich

```
POKE 54296,15
```

eingegeben werden.

Dadurch wird die Lautstärke im SID auf 15 gesetzt. Vergessen Sie aber nicht, anschließend die Lautstärke des Lautsprechers auf max. zu drehen. Wenn nun geladen, geSAVEd, oder VERIFY ausgeführt wird, so können Sie die Cassette laut und klar hören.

Zum einen erhalten Sie durch diese Mithörmöglichkeit die Gelegenheit, eventuelle Fehlerquellen schnell zu erkennen.

Sie können dies jedoch auch einsetzen, um eine Musikcassette abzuspielen. Da jedoch durch diesen POKE zwar die Frequenz des Signals, nicht jedoch die Tonqualtiät originalgetreu wieder-

gegeben wird, sollten Sie an die Qualität der wiedergegebenen Titel nicht allzu hohe Anforderungen stellen.

Mit POKE 54296,0 schalten Sie den Audio-Sound wieder ab.

3.1.2 Sound-Filter

Mittels dieses Filters können Sie statische Rauschgeräusche abschalten.

Ergänzen Sie Ihr eigenes Sound-/Musikprogramm dafür mit dem Befehl

```
POKE 54295,4
```

Dadurch wird das Bit 4 des Resonanz-Filters aktiviert. Dies bewirkt, daß alles unerwünschte Rauschen ausgeblendet wird. Zurück bleibt der klare, saubere Ton der gerade ausgegebenen Note.

Der Poke muß dabei eingefügt werden, nachdem die Sound-Register gelöscht worden sind.

3.1.3 Sound-Verbesserung

Eine wesentliche Verbesserung der Tonausgabe können Sie auch über das Abschalten des Bildschirms während der Ausgabe erreichen.

Wie bereits erläutert geschieht dies mittels

```
POKE 53265,11 .
```

Das Wiedereinschalten des Displays benötigt den Befehl:

```
POKE 53265,27
```

Diese Maßnahme verbessert im wesentlichen den Rhythmus eines Sounds oder einer Melodie.

Dies liegt daran, daß der Computer bei einem abgeschalteten Bildschirm diesen nicht mehr ständig wieder neu aufbauen muß. Da das ständige Neuerstellen immer zwischen den einzelnen Noten erfolgt, beinträchtigt dies oft den Rhythmus.

Duch das Abschalten des Displays erhalten Sie so nicht nur einen saubereren und gleichmäßigen Rhytmus, sondern gleichzeitig erhöht sich auch die Geschwindigkeit der Tonabfolge.

3.1.4 Sound-Dämmung

Der Datasette-Audio-Poke läßt sich auch dazu verwenden, zu verhindern, daß ein Ton auf einer geringen Lautstärke weiterertönt, obwohl er eigentlich beendet sein sollte.

Geben Sie in diesem Fall POKE 54296,0 nach jedem Ton ein und POKE 54296,15 für die maximale Lautstärke des SIDs.

3.1.5 Soundroutinen

Nun folgt eine kleine Sammlung einzelner Sound-Effekte. Diese können Sie direkt zur Animation in Ihre eigenen Programme integrieren oder aber beliebig mit Ihnen experimentieren, um so neue Effekte zu generieren.

SOUND NR.1:

```
5 REM SOUND 1
10 FOR I=54272 TO 54295:POKE I,0:NEXT I
20 POKE 54296,15
30 POKE 54277,8
40 POKE 54278,255
42 POKE 54276,23
45 FOR Z=1 TO 5
50 A=2:FOR Y=1 TO 3
60 B=8:POKE 54273,A
70 FOR I=1 TO 6
75 POKE 54287,B
80 B=B*1.4:NEXT I
90 A=A*1.5:NEXT Y
100 FOR I=1 TO 10
105 POKE 54287,B
110 B=B*1.05
115 NEXT I
120 NEXT Z
130 POKE 54278,15
```

SOUND NR.2:

```
5 REM SOUND 2
10 FOR I=54272 TO 54295:POKE I,0:NEXT I
20 POKE 54296,15
30 POKE 54277,8
35 POKE 54278,255
40 POKE 54276,21
45 FOR K=1 TO 4
50 A=20:FOR Y=1 TO 2
60 B=90:POKE 54273,A
70 FOR I=1 TO 6
75 POKE 54287,B
80 B=B*.3
90 A=A*1.1
100 NEXT I
110 NEXT Y
120 NEXT K
130 POKE 54278,15
```

SOUND NR.3:

```
5 REM SOUND 3
10 FOR I=54272 TO 54295:POKE I,0:NEXT
20 POKE 54296,15
30 POKE 54277,8
35 POKE 54278,255
40 POKE 54276,23
45 FOR Z=1 TO 3
50 A=9:FOR Y=1 TO 6
60 B=30
70 FOR X=1 TO 3:POKE 54287,B
80 B=B/1.5:NEXT X
90 NEXT Y
92 FOR I=1 TO 10:POKE 54273,A
94 POKE 54287,5
120 NEXT I
130 POKE 54278,15
```

SOUND NR.4:

```
5 REM SOUND 4
10 FOR I=54272 TO 54295:POKE I,0:NEXT I
20 POKE 54296,15
30 POKE 54277,8
35 POKE 54278,255
40 POKE 54276,21
50 FOR I=1 TO 6
60 FOR A=255 TO 0 STEP -2
70 POKE 54273,A
80 NEXT A
90 NEXT I
95 POKE 54278,15
```

SOUND NR.5:

```
5 REM SOUND 5
10 FOR I=54272 TO 54295:POKE I,0:NEXT
20 POKE 54296,15
35 POKE 54278,255
40 POKE 54276,23
45 POKE 54287,25:REM TONART
50 FOR I=1 TO 6
60 FOR A=255 TO 0 STEP -4
70 POKE 54273,A
80 NEXT A
90 NEXT I
95 POKE 54278,15
```

SOUND NR.6:

```
5 REM SOUND 6
10 FOR I=54272 TO 54295:POKE I,0:NEXT I
20 POKE 54296,15
30 POKE 54278,255
35 POKE 54277,9
40 POKE 54287,25
45 POKE 54276,21
50 FOR I=1 TO 6
60 FOR A=255 TO 0 STEP -4
70 POKE 54273,A
75 NEXT A
80 FOR A=0 TO 255 STEP 6
85 POKE 54273,A
87 NEXT A
90 NEXT I
95 POKE 54278,15
```

SOUND NR.7:

```
5 REM SOUND 7
10 FOR I=54272 TO 54295:POKE I,0:NEXT
30 POKE 54277,8
40 POKE 54278,225
43 POKE 54276,23
45 FOR I=1 TO 3
50 A=50:FOR Y=1 TO 5
60 B=8:POKE 54273,A
70 FOR X=1 TO 9:POKE 54287,B
80 B=B*1.2:NEXT X:A=A*1.5
120 NEXT Y:NEXT I
130 POKE 54278,15
```

SOUND NR.8:

```
10 REM SOUND 8
20 CH=54272
30 FOR T=CH TO CH+24:POKE T,0:NEXT
35 INPUT "RYTHMUS-RATE 0-15";RR
40 INPUT "ABSCHWELL-RATE:TON 0-15";AT
50 SU=15
60 INPUT "ABSCHWELL-RATE:GESAMT 0-15";AG
70 POKE 54267,23
80 POKE 54296,15:POKE 54277,16*RR+AT
90 POKE 54278,240+AG
95 :
100 FOR T=10 TO 90 STEP 10:POKE 54276,17
110 :    POKE 54272,50:POKE 54273,T
120 :    FOR J=1 TO 180:NEXT J
125 :    POKE 54276,16
130 :    FOR H=1 TO 2↑AG:NEXT H
140 NEXT T:PRINT "ECHO"
```

```
150 FOR T=80 TO 10 STEP -9
160 :  FOR DB=10 TO 2 STEP -1/3
180 :  POKE 54276,17:POKE 54296,DB
190 :  POKE 54273,T:FOR J=1 TO 7:NEXT J
200 POKE 54273,100-T:FORJ=1 TO 7:NEXT J
210 NEXT DB:NEXT T:POKE 54276,15
```

SOUND NR.9:

```
20 REM SOUND 9
30 FOR T=54272 TO 54296:POKE T,0:NEXT
70 POKE 54267,23
80 POKE 54296,15:POKE 54277,16+5
90 POKE 54278,255
95 :
100 FOR T=10 TO 90 STEP 10:POKE 54276,17
110 :   POKE 54272,50:POKE 54273,T
120 :   FOR I=1 TO 200:NEXT I
125 :   POKE 54276,16
130 :   FOR H=1 TO 32 :NEXT H
140 NEXT T:PRINT "ECHO"
150 FOR T=90 TO 10 STEP -7
160 : FOR K=12 TO 2 STEP -1/3
180 :  POKE 54276,17:POKE 54296,K
190 :  POKE 54273,T:FOR P=1 TO 5:NEXT P
200 :  POKE 54273,100-T:FORJ=1TO8:NEXTJ
210 : NEXT K
220 NEXT T:POKE 54276,15
```

SOUND NR.10:

```
20 REM SOUND 10
30 FOR T=54272 TO 54296:POKE T,0:NEXT
70 POKE 54267,23
80 POKE 54296,15:POKE 54277,16+5
90 POKE 54278,255
95 :
100 POKE 54276,17
```

```
110 :    POKE 54272,50
120 FOR I=1 TO 5:FOR K=255TO1STEP-6:POKE54
273,K:NEXT K
125 FOR K=1 TO 255 STEP 16:POKE 54273,K:NE
XT K:NEXT I
130 :
140 PRINT "ECHO"
150 FOR T=255 TO 1 STEP -20
160 : FOR K=12 TO 2 STEP -1/3
180 POKE 54296,K
190 :  POKE 54273,T:FOR J=1 TO 7:NEXT J
200 :  POKE 54273,255-T:FORJ=1 TO 7:NEXT J
210 : NEXT K
220 NEXT T:POKE 54276,15
```

SOUND NR.11:

```
5 REM SOUND 11
10 FOR I=54272 TO 54295:POKE I,0:NEXT
20 POKE 54296,15
30 POKE54277,8
40 POKE 54278,255
42 POKE 54276,23
43 :
45 FOR A=1 TO 3:D=2
50 FOR B=1 TO 3:G=4
60 POKE 54273,D
70 FOR C=4 TO 30:POKE 54287,C
80 G=G*1.1:POKE 54273,G:NEXT
90 D=D*1.5:NEXT B
100 FOR W=1 TO 10:POKE 54287,E
110 E=E*1.1:NEXT W
120 NEXT A:POKE 54278,15
```

3.2 Synthesizer in Stereo

Wenn Sie des öfteren mit dem Synthesizer Ihres C64 gearbeitet haben, werden Sie sich auch sicher schon über den etwas mageren Klang geärgert haben, den Ihr Fernsehgerät produziert. Das mag auch von der Qualität des Tonteiles im Fernseher ab

hängen. Eine evtl. vorhandene Stereoanlage fordert einen geradezu heraus, hier Abhilfe zu schaffen und einen etwas volleren Sound zu produzieren.

Da eine Stereoanlage bekanntlich über zwei Kanäle verfügt, müssen wir uns zunächst Gedanken darüber machen, wie wir den einkanaligen Ausgang des Synthesizers auf diese beiden Kanäle aufteilen.

Leider sind die einzelnen Stimmen des Gerätes nicht an separaten Anschlüssen verfügbar, sonst wäre eine Aufteilung schon gegeben.

Also sind wir hingegangen und haben das Tonsignal in zwei Frequenzbereiche aufgeteilt. Die Aufteilung in unteren und oberen Bereich geschieht bei 300 Hz.

Damit wird der vom Synthesizer überstrichene Bereich gehörrichtig in der Mitte geteilt, nämlich in drei Oktaven unterhalb (bis 36 Hz) und vier Oktaven oberhalb (bis 4800 Hz) 300 Hz.

Realisiert wird dies mit zwei Doppel-T-Filtern mit einer Flankensteilheit von 6dB/Oktave und einer Grenzfrequenz von 300 Hz (Tiefpaß) bzw. 3 kHz (Hochpaß).

Die Grenzfrequenzen können Sie natürlich nach Geschmack durch Verwendung anderer Kondensatoren verändern, jedoch sollten Sie die Widerstandswerte beibehalten, da diese auf die Impedanz der angeschlossenen Geräte abgestimmt sind.

Bei gegebener Grenzfrequenz F errechnet sich der benötigte Kondensatorwert mit C=1/(3300*F).

Haben sie irgendwelche Kondensatoren in Ihrer Bastelkiste und wollen wissen, welche Grenzfrequenz sich daraus ergibt, so ist die Formel F=1/(3300*C) anzuwenden.

Sie können allerdings davon ausgehen, daß die Schaltung mit den angegebenen Werten ein Optimum darstellt, das wir in unserem Labor durch ausgiebige Messungen und Hörtests erreicht haben.

Genügt Ihnen eine Flankensteilheit des Filters von 3dB/Oktave, so können Sie die Bauteile R2, C2, C4, R6 auch weglassen. Dabei wird der Klang über einen weiteren Bereich akustisch zwischen den Lautsprechern stehen.

Wie Sie sehen, ist die Filterschaltung extrem einfach. Sie wird zweckmäßig auf einem kleinen Stück Lochrasterplatte aufgebaut und in die Verbindungsleitung zur Stereoanlage eingeschleift. Wer ein Übriges tun will, kann diese Anordnung mitsamt der angeschlossenen Kabelenden in Silikonkautschuk eingießen. Damit erhalten Sie eine Lösung, die auch optisch einwandfrei ist.

Wir wollen Ihnen nun ein kurzes Programm vorstellen, das einen 'Sweep' mit der Dreieckschwingung produziert.

Hierbei können Sie deutlich hören, wie der Ton von einem Lautsprecher zum anderen wandert.

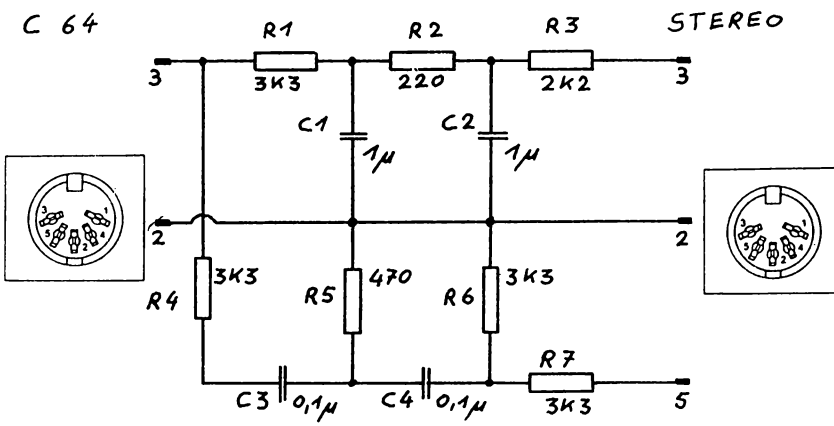
Wir haben die Dreieckschwingung deshalb gewählt, weil sie relativ obertonarm ist und Sie deshalb den Effekt besser beobachten können.

Bei komplexeren, obertonreichen Klängen, z.B. Sägezahnschwingung, erscheinen die Obertöne auf dem einen Kanal, während Sie die Grundschwingung aus dem anderen Lautsprecher hören, sofern diese 300 Hz nicht übersteigt.
Eines werden Sie aber sicher feststellen, daß nämlich der Klang wesentlich voluminöser geworden ist.

Hier nun das Programm:

```
10 s1=54272
20 s2=54279
30 s3=54286
60 rs=54295
70 pl=54296
80  poke s1+4,0:poke s2+4,0:poke s3+4,0
100 a=9:d=9:s=9:r=9:h=30
110 poke rs,0:poke pl,15
120 poke s3+5,16*a+d:poke s3+6,16*s+r
130 poke s3+4,17
140 for i=0 to h:poke s3+1,peek(54300):next i
150 poke s3+4,16
160 for i=0 to r*4:poke s3+1,peek(54300):next i
```

Auf die untenstehende Schaltung gehen wir ihrer Einfachheit wegen nicht näher ein. Die stereoseitige Steckerbelegung ist zum Anschluß an den Phonoeingang gedacht.



3.3 Sounderweiterungen für den C-64

Leider ist es auch bei den Soundmöglichkeiten des C-64 so, daß diese nicht durch Befehle unterstützt werden. Besonders aufwendig, für Anfänger fast unmöglich, ist es dann, ganze 3-stimmige Lieder zu spielen. Nun ist es aber auch möglich, eigene Befehle zu programmieren, die dann ganz einfach durch einen erweiterten SYS-Befehl (siehe Kapitel 6.1) aufgerufen werden. Eine solche Sammlung von Routinen haben wir nun für Sie im folgenden erstellt. Diese Routinen, deren Funktionen jeweils im Anschluß an die entsprechenden Programmlistings erklärt sind, können Sie dann, ganz nach Bedarf in Ihre eigenen Programme aufnehmen:

Der VOLUME-Befehl

Zunächst das Assemblerlisting:

```
110:  c000                    .opt p,oo
120:  c000                    *=   49152
130:                ;
140:                ;******************************
150:                ;*** vol ***********************
160:                ;******************************
170:                ;
180:  c000 20 fd ae vol       jsr  $aefd
180:  c003 20 9e b7           jsr  $b79e
180:  c006 e0 10              cpx  #16
180:  c008 90 03              bcc  vok1
190:  c00a 4c 11 c0           jmp  volerr
200:  c00d 8e 18 d4 vok1      stx  54272+24
210:  c010 60                 rts
220:                ;
```

```
230:   c011 a9 1c    volerr   lda  #<vetext
240:   c013 a0 c0             ldy  #>vetext
250:   c015 85 22             sta  $22
250:   c017 84 23             sty  $23
260:   c019 4c 4a a4          jmp  $a44a
270:                ;
280:   c01c 56 4f 4c vetext   .asc "volume too higH"
```

Dazu nun der BASIC-Lader:

```
10 rem ********************************
20 rem ***                          ***
30 rem ***    basiclader "vol"      ***
40 rem ***   ------------------     ***
50 rem ***                          ***
60 rem ********************************
70 :
80 for x=49152 to 49168
90 :   read a
100 :  poke x,a
110 :  p=p+a
120 next x
130 :
140 if p<> 1978 then print "fehler in datas !!!":stop
150 :
160 end
170 :
5000 data  32,253,174, 32,158,183,224
5010 data  16,144,  3, 76, 17,192,142
5020 data  24,212, 96
```

Die Anwendung dieses kleinen Assemblermoduls ist nun denkbar einfach. Sie müssen einfach diese Erweiterung mit

```
SYS 49152,vol
```

ausrufen und schon wird das Volumen durch den Wert, der in das Register 24 geschrieben wird, auf Ihr gewünschtes Level gesetzt.

Envelope

Zunächst wieder das Assemblerlisting:

```
110:   c020                    .opt p1,oo
120:   c020                    *=   $c020
130:                  ;
140:                  ;*********************************
150:                  ;*** envelope *********************
160:                  ;*********************************
170:                  ;
180:   c020 4c 4f c0           jmp  envel
190:   c023 98        getnib   tya
190:   c024 48                 pha
190:   c025 20 fd ae           jsr  $aefd
200:   c028 20 9e b7           jsr  $b79e
200:   c02b e0 10              cpx  #16
200:   c02d 90 03              bcc  eok1
200:   c02f 4c a8 c0           jmp  ill2
210:   c032 8a        eok1     txa
210:   c033 0a                 asl
210:   c034 0a                 asl
210:   c035 0a                 asl
210:   c036 0a                 asl
210:   c037 8d b3 c0           sta  ez1
220:   c03a 20 fd ae           jsr  $aefd
220:   c03d 20 9e b7           jsr  $b79e
```

```
220:  c040 68              pla
220:  c041 a8              tay
220:  c042 8a              txa
220:  c043 e0 10           cpx  #16
230:  c045 90 03           bcc  eok2
230:  c047 4c a8 c0        jmp  ill2
240:  c04a 18       eok2   clc
240:  c04b 6d b3 c0        adc  ez1
240:  c04e 60              rts
250:                ;
260:  c04f 20 fd ae envel  jsr  $aefd
260:  c052 20 9e b7        jsr  $b79e
260:  c055 e0 00           cpx  #0
270:  c057 d0 03           bne  eok4    ; "groesser als 0 ?"
280:  c059 4c 75 c0        jmp  ill
290:  c05c e0 04    eok4   cpx  #4      ;"kleiner als 3 ?"
300:  c05e 90 03           bcc  eok3    ;"jawohl !"
310:  c060 4c 75 c0        jmp  ill
320:                ;
330:  c063 ca       eok3   dex
330:  c064 bd b5 c0        lda  stimmen,x
330:  c067 a8              tay
340:  c068 20 23 c0        jsr  getnib
340:  c06b 99 05 d4        sta  54272+5,y
350:  c06e 20 23 c0        jsr  getnib
350:  c071 99 06 d4        sta  54272+6,y
350:  c074 60              rts
360:                ;
370:  c075 a9 80    ill    lda  #<ivn
370:  c077 a0 c0           ldy  #>ivn
380:  c079 85 22    error  sta  $22
380:  c07b 84 23           sty  $23
380:  c07d 4c 47 a4        jmp  $a447
390:                ;
400:  c080 49 4c 4c ivn    .asc "illegal voice numbeR"
410:  c094 45 4e 56 enth   .asc "envelnumber too higH"
420:  c0a8 a9 94    ill2   lda  #<enth
420:  c0aa a0 c0           ldy  #>enth
```

```
430:   c0ac 85 22            sta  $22
430:   c0ae 84 23            sty  $23
430:   c0b0 4c 47 a4         jmp  $a447
440:                ;
450:   c0b3 00       ez1     .byt 0
460:   c0b4 00       gd1     .byt 0
470:                ;
480:   c0b5 00 07 0e stimmen  .byt 0,7,14
```

Und wieder der BASIC-Lader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***  basiclader "envelope"     ***
130 rem *** ---------------------      ***
140 rem ***                            ***
150 rem ********************************
160 :
170 for x=49184 to 49336
180 :    read a
190 :    poke x,a
200 :    p=p+a
210 next x
220 :
230 if p<> 16067 then print "fehler in datas !!!":stop
240 :
250 end
260 :
5000 data  76, 79,192,152, 72, 32,253
5010 data 174, 32,158,183,224, 16,144
5020 data   3, 76,168,192,138, 10, 10
5030 data  10, 10,141,179,192, 32,253
5040 data 174, 32,158,183,104,168,138
5050 data 224, 16,144,  3, 76,168,192
5060 data  24,109,179,192, 96, 32,253
5070 data 174, 32,158,183,224,  0,208
5080 data   3, 76,117,192,224,  4,144
```

```
5090 data   3, 76,117,192,202,189,181
5100 data 192,168, 32, 35,192,153,  5
5110 data 212, 32, 35,192,153,  6,212
5120 data  96,169,128,160,192,133, 34
5130 data 132, 35, 76, 71,164, 73, 76
5140 data  76, 69, 71, 65, 76, 32, 86
5150 data  79, 73, 67, 69, 32, 78, 85
5160 data  77, 66, 69,210, 69, 78, 86
5170 data  69, 76, 78, 85, 77, 66, 69
5180 data  82, 32, 84, 79, 79, 32, 72
5190 data  73, 71,200,169,148,160,192
5200 data 133, 34,132, 35, 76, 71,164
5210 data   0,  0,  0,  7, 14, 72
```

Nun zur Erklärung des Envelope-Befehls. Dieser Befehl setzt die Hüllkurve einer beliebigen Stimme. Der Aufruf lautet

```
SYS 49184,stimme,attack,decay,sustain,release
```

wobei die verwendeten Variablen folgende Bedeutung haben:

stimme	Hier ist die Stimme (1-3) anzugeben, für die die Hüllkurve bestimmt werden soll. Wird die Hüllkurve einer nicht-existierenden Stimme gesetzt, so wird die Fehlermeldung "ILLEGAL VOICE NUMBER ERROR" ausgegeben.
attack	Durch diese Variable wird der Anschlag (0-15) bestimmt. Werden die zulässigen Werte überschritten, so wird die Fehlermeldung "ENVEL NUMBER TOO HIGH

ERROR" ausgegeben. Dies gilt auch für alle nachfolgenden Parameter.

decay, sustain, release — Durch diese Parameter wird der Rest der Hüllkurve äquivalent zu 'attack' gesetzt.

Wave

Das Assemblerlisting:

```
110:   c100                    .opt p1,oo
120:   c100                    *=   $c100
130:                  ;
140:                  ;*********************************
150:                  ;*** wave ***********************
160:                  ;*********************************
170:                  ;
180:   c100 20 fd ae wave      jsr  $aefd
180:   c103 20 9e b7           jsr  $b79e
190:   c106 20 71 c1           jsr  chek
190:   c109 8e 4a c1           stx  wzw
190:   c10c a9 00              lda  #0
200:   c10e 8d 4b c1           sta  wzw+1
200:   c111 20 fd ae           jsr  $aefd
210:   c114 20 6a c1           jsr  getstr
220:   c117 c9 08              cmp  #8
220:   c119 f0 03              beq  wavok1
230:   c11b 4c 9f c1           jmp  ill3
240:   c11e a0 00     wavok1   ldy  #0
250:   c120 b1 22     wavlab   lda  ($22),y
250:   c122 c9 30              cmp  #"0"
250:   c124 f0 10              beq  wavw1
260:   c126 c9 31              cmp  #"1"
260:   c128 f0 03              beq  wavw2
270:   c12a 4c ba c1           jmp  ill4
280:   c12d b9 4c c1 wavw2     lda  wavtab,y
```

```
290:   c130 0d 4b c1          ora  wzw+1
290:   c133 8d 4b c1          sta  wzw+1
300:   c136 c8        wavw1   iny
300:   c137 c0 08             cpy  #8
300:   c139 90 e5             bcc  wavlab
310:   c13b ae 4a c1          ldx  wzw
310:   c13e ca                dex
310:   c13f bd d2 c1          lda  stimmen,x
320:   c142 aa                tax
320:   c143 ad 4b c1          lda  wzw+1
330:   c146 9d 04 d4          sta  54272+4,x
340:   c149 60                rts
350:                  ;
360:   c14a 00 00     wzw      .byt 0,0
370:   c14c 80 40 20 wavtab    .byt 128,64,32,16,8,4,2,1
380:                  ;
390:   c154 a9 5b     ill5     lda  #<ilv
390:   c156 a0 c1              ldy  #>ilv
390:   c158 4c 84 c1           jmp  skip
400:                  ;
410:   c15b 56 4f 4c ilv       .asc "volume too higH"
420:                  ;
430:   c16a 18        getstr   clc
440:   c16b 20 9e ad           jsr  $ad9e
440:   c16e 4c a3 b6           jmp  $b6a3
450:                  ;
460:   c171 e0 00     chek     cpx  #0
470:   c173 d0 03              bne  chkok1
480:   c175 4c 80 c1           jmp  ill
490:   c178 e0 04     chkok1   cpx  #4
500:   c17a 90 03              bcc  chkok2
510:   c17c 4c 80 c1           jmp  ill
520:   c17f 60        chkok2   rts
530:                  ;
540:   c180 a9 8b     ill      lda  #<ivn
540:   c182 a0 c1              ldy  #>ivn
550:   c184 85 22     skip     sta  $22
550:   c186 84 23              sty  $23
560:   c188 4c 47 a4           jmp  $a447
570:                  ;
```

```
580:   c18b 49 4c 4c ivn      .asc "illegal voice numbeR"
590:                ;
600:   c19f a9 a6    ill3     lda  #<ise
600:   c1a1 a0 c1             ldy  #>ise
610:   c1a3 4c 84 c1          jmp  skip
620:                ;
630:   c1a6 49 4c 4c ise      .asc "illegal stringlenghT"
640:                ;
650:   c1ba a9 c1    ill4     lda  #<wce
650:   c1bc a0 c1             ldy  #>wce
660:   c1be 4c 84 c1          jmp  skip
670:                ;
680:   c1c1 49 4c 4c wce      .asc "illegal characteR"
690:                ;
700:   c1d2 00 07 0e stimmen  .byt 0,7,14
```

Und der BASIC-Lader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***    basiclader "wave"       ***
130 rem ***   ------------------       ***
140 rem ***                            ***
150 rem ********************************
160 :
170 for x=49408 to 49621
180 :   read a
190 :   poke x,a
200 :   p=p+a
210 next x
220 :
230 if p<> 23029 then print "fehler in datas !!!":end
240 :
250 end
260 :
```

```
5000 data  32,253,174, 32,158,183, 32
5010 data 113,193,142, 74,193,169,  0
5020 data 141, 75,193, 32,253,174, 32
5030 data 106,193,201,  8,240,  3, 76
5040 data 159,193,160,  0,177, 34,201
5050 data  48,240, 16,201, 49,240,  3
5060 data  76,186,193,185, 76,193, 13
5070 data  75,193,141, 75,193,200,192
5080 data   8,144,229,174, 74,193,202
5090 data 189,210,193,170,173, 75,193
5100 data 157,  4,212, 96,  0,  0,128
5110 data  64, 32, 16,  8,  4,  2,  1
5120 data 169, 91,160,193, 76,132,193
5130 data  86, 79, 76, 85, 77, 69, 32
5140 data  84, 79, 79, 32, 72, 73, 71
5150 data 200, 24, 32,158,173, 76,163
5160 data 182,224,  0,208,  3, 76,128
5170 data 193,224,  4,144,  3, 76,128
5180 data 193, 96,169,139,160,193,133
5190 data  34,132, 35, 76, 71,164, 73
5200 data  76, 76, 69, 71, 65, 76, 32
5210 data  86, 79, 73, 67, 69, 32, 78
5220 data  85, 77, 66, 69,210,169,166
5230 data 160,193, 76,132,193, 73, 76
5240 data  76, 69, 71, 65, 76, 32, 83
5250 data  84, 82, 73, 78, 71, 76, 69
5260 data  78, 71, 72,212,169,193,160
5270 data 193, 76,132,193, 73, 76, 76
5280 data  69, 71, 65, 76, 32, 67, 72
5290 data  65, 82, 65, 67, 84, 69,210
5300 data   0,  7, 14,208
```

Auch dieser Befehl ist wieder sehr nützlich. Er setzt nämlich auf komfortable Weise das Steuerregister des SIDs, indem die Wellenformen, die Ringmodulation, die Synchronisation, das Testbit und das Hüllkurvenbit gesteuert werden. Statt nun umständlich eine Zahl in dieses Register 4 zu POKEn, brauchen Sie

einfach nur das entsprechende Bit-Muster in einem String zu übergeben. Der Aufruf der Routine lautet nun:

```
SYS 49408,stimme,"wert"
```

Wie schon gesagt, wird bei diesem Befehl aus dem Bit-Muster des Strings der entsprechnde Wert errechnet und dieser in das Steuerregister geschrieben. Als Beispiel: Sie möchten die Dreicks-Wellenform einschalten und die Hüllkurve der Stimme 1 starten. Also schreiben Sie nur

```
SYS 49408,1,"00010001"
```

und schon werden alle gewünschten Befehle ausgeführt.

Sdata

Das Assemblerlisting zu SDATA:

```
110:   c200                    .opt p1,oo
120:   c200                    *=   $c200
130:                  ;
140:                  ;*********************************
150:                  ;*** sdata ************************
160:                  ;*********************************
170:                  ;
180:   c200 20 fd ae sdata     jsr  $aefd
180:   c203 20 9e b7           jsr  $b79e
180:   c206 8e c8 c2           stx  rpz
190:   c209 20 fd ae           jsr  $aefd
190:   c20c 20 be c2           jsr  getstr
200:   c20f c9 01              cmp  #1
200:   c211 f0 1f              beq  sdok1
```

```
210:   c213 a9 1e    ill3     lda  #<i3
210:   c215 a0 c2             ldy  #>i3
220:   c217 85 22             sta  $22
220:   c219 84 23             sty  $23
220:   c21b 4c 47 a4          jmp  $a447
230:                 ;
240:   c21e 49 4c 4c i3       .asc "illegal stringlengtH"
250:   c232 a2 00    sdok1    ldx  #0
260:   c234 a0 00             ldy  #0
270:   c236 b1 22             lda  ($22),y
280:   c238 dd 79 c2 sdlab    cmp  nntab,x
290:   c23b f0 20             beq  found2
300:   c23d e8                inx
300:   c23e e0 0c             cpx  #12
300:   c240 90 f6             bcc  sdlab
310:   c242 a9 4d    ill4     lda  #<i4
310:   c244 a0 c2             ldy  #>i4
320:   c246 85 22    ea2      sta  $22
320:   c248 84 23             sty  $23
320:   c24a 4c 47 a4          jmp  $a447
330:                 ;
340:   c24d 57 52 4f i4       .asc "wrong stringnam"
340:   c25c c5                .byt "e"+$80
350:                 ;
360:   c25d 8a        found2   txa
360:   c25e 0a                asl
360:   c25f aa                tax
370:   c260 bd 87 c2          lda  nwtab,x
370:   c263 85 14             sta  $14
370:   c265 bd 88 c2          lda  nwtab+1,x
380:   c268 85 15             sta  $15
390:   c26a ae c8 c2          ldx  rpz
390:   c26d f0 07             beq  sdend1
400:   c26f 06 14    sdl2     asl  $14
400:   c271 26 15             rol  $15
400:   c273 ca                dex
400:   c274 d0 f9             bne  sdl2
410:   c276 4c 9f c2 sdend1   jmp  ubvar
420:                 ;
```

```
430:   c279 43 03 44 nntab    .byt 67,3,68,4,69,5,70,6,71
440:   c282 07 41 01          .byt 7,65,1,72,8
450:   c287 16 01     nwtab   .wor 278
460:   c289 27 01 39          .wor 295,313,331,351,372,394,417
470:   c297 ba 01 d4          .wor 442,468,496,526
480:                  ;
490:   c29f 20 fd ae ubvar    jsr  $aefd
490:   c2a2 20 8b b0          jsr  $b08b
500:   c2a5 a5 14             lda  $14
500:   c2a7 85 63             sta  $63
500:   c2a9 a5 15             lda  $15
500:   c2ab 85 62             sta  $62
510:   c2ad a2 90             ldx  #$90
510:   c2af 38                sec
510:   c2b0 20 49 bc          jsr  $bc49
520:   c2b3 a6 47             ldx  $47
530:   c2b5 86 49             stx  $49
540:   c2b7 a6 48             ldx  $48
550:   c2b9 86 4a             stx  $4a
560:   c2bb 4c d0 bb          jmp  $bbd0
570:                  ;
580:                  ;*** get string ***
590:                  ;
600:   c2be 18       getstr   clc
610:   c2bf 20 9e ad          jsr  $ad9e
620:   c2c2 20 8f ad          jsr  $ad8f
630:   c2c5 4c a6 b6          jmp  $b6a6
640:                  ;
650:   c2c8 00       rpz      .byt 0
```

Und der BASIC-Lader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***   basiclader "sdata"       ***
130 rem ***  ----------------------    ***
140 rem ***                            ***
150 rem ********************************
```

```
160 :
170 for x=49664 to 49865
180 :   read a
190 :   poke x,a
200 :   p=p+a
210 next x
220 :
230 if p<> 20819 then print "fehler in datas !!!":stop
240 :
250 end
260 :
5000 data  32,253,174, 32,158,183,142
5010 data 200,194, 32,253,174, 32,190
5020 data 194,201,  1,240, 31,169, 30
5030 data 160,194,133, 34,132, 35, 76
5040 data  71,164, 73, 76, 76, 69, 71
5050 data  65, 76, 32, 83, 84, 82, 73
5060 data  78, 71, 76, 69, 78, 71, 84
5070 data 200,162,  0,160,  0,177, 34
5080 data 221,121,194,240, 32,232,224
5090 data  12,144,246,169, 77,160,194
5100 data 133, 34,132, 35, 76, 71,164
5110 data  87, 82, 79, 78, 71, 32, 83
5120 data  84, 82, 73, 78, 71, 78, 65
5130 data  77,197,138, 10,170,189,135
5140 data 194,133, 20,189,136,194,133
5150 data  21,174,200,194,240,  7,  6
5160 data  20, 38, 21,202,208,249, 76
5170 data 159,194, 67,  3, 68,  4, 69
5180 data   5, 70,  6, 71,  7, 65,  1
5190 data  72,  8, 22,  1, 39,  1, 57
5200 data   1, 75,  1, 95,  1,116,  1
5210 data 138,  1,161,  1,186,  1,212
5220 data   1,240,  1, 14,  2, 32,253
5230 data 174, 32,139,176,165, 20,133
5240 data  99,165, 21,133, 98,162,144
5250 data  56, 32, 73,188,166, 71,134
```

```
5260 data  73,166,  72,134,  74,  76,208
5270 data 187,  24,  32,158,173,  32,143
5280 data 173,  76,166,182,   0,208,  39
```

Und nun wieder die Erklärung. Dieser Befehl errechnet Ihnen aus der angegebenen Oktave und dem Notenstring den Wert, den Sie in die Register 0,1 ; 7,8 oder 14,15 schreiben müssen, damit Ihre Note die richtige Tonhöhe hat. Das erspart 1. aufwendiges Blättern im Handbuch und ist 2. komfortabler zu programmieren. Der Aufruf des Befehls lautet:

```
SYS 49664,oktave,"note",variable
```

wobei 'oktave' die entsprechende Oktave, und "note" ein String ist, in dem ein Notensymbol in Form eines Buchstabens stehen muß. Soll diese Note um einen Halbton erhöht werden, so wird der entsprechende Buchstabe zusammen mit der Control-Taste betätigt.

'variable' ist eine beliebige Variable, in der der errechnete Wert übergeben wird.

4. Kapitel

Benutzerschnittstellen

4.1 Maussimulation mit dem Joystick

Ein neues Schlagwort hat die Welt der Personalcomputer heimgesucht: Die "MAUS".

Was verbirgt sich hinter diesem für die Branche etwas ungewöhnlichen Ausdruck?

Sicher kennen Sie von den Videospielautomaten den sogenannten Spinball. Dieses ist, an Stelle von Joysticks oder Paddles, ein Betätigungselement, mit dem Sie irgendwelche Figuren über den Bildschirm bewegen können.

Im Gegensatz zu Joysticks, deren Hebel Sie nur in acht Richtungen bewegen können, erlaubt der Spinball ein Drehen in alle Richtungen, da es sich hierbei um eine frei gelagerte Kugel handelt, deren Bewegung auf zwei Winkelcodierer übertragen wird, und zwar auf je einen für die X- und die Y-Achse.

Bei den Spielautomaten wird diese Kugel in der Regel mit der Handfläche in Bewegung gesetzt.

Die "Maus" besteht aus einem ebensolchen Spinball, jedoch ist dieser in ein handliches Gehäuse eingebaut, das man, Kugel zuunterst, auf den Tisch legt und nun mit der Hand hin- und herrollen kann. Durch das Bewegen des Gehäuses auf dem Tisch wird die Kugel durch die Reibung auf der Tischfläche in Rotation versetzt. Durch ein geeignetes Interface wird der jeweilige Stand der beiden Winkelcodierer in den Rechner übertragen.

Die sinnige Bezeichnung dieser Anordnung mag wohl aus der Ähnlichkeit mit einer Aufziehmaus herrühren.

Wie wendet man nun die Maus in durchaus als ernsthaft zu bezeichnenden Programmen an?

Dazu müssen wir ein wenig ausholen:

Wenn ein Programm, das einem breiten Anwenderkreis zugänglich ist, das Prädikat 'bedienungsfreundlich' zu Recht tragen

soll, wird es in aller Regel nach der Menütechnik aufgebaut sein. Dieses Verfahren hat den Vorteil, daß es nahezu von jedem Bediener gehandhabt werden kann.

Hierbei werden die vom Anwender verlangten Aktivitäten diesem, entweder einzeln oder zur Auswahl aus einer Gruppe von Möglichkeiten, über den Bildschirm mitgeteilt.

Die Auswahl erfolgt entweder durch Eingeben eines Kennbuchstabens oder durch Bewegen des Cursor in ein zu diesem Zweck am Bildschirm aufgebautes Feld.

Nun haben Fachleute, die etwas von Ergonomie verstehen, herausgefunden, daß der Vorgang der Auswahl bequemer und auch sicherer erfolgen kann, wenn nicht auf der Tastatur nach dem Adlersystem der passende Buchstabe herausgesucht werden muß, sondern das Ganze im weichen Sessel zurückgelehnt geschehen kann.

Das setzt natürlich voraus, daß das gesamte Programm derart gehandhabt werden kann, d.h. sämtliche erforderlichen Eingaben müssen auf dieses System abgestimmt sein.

Bei der Maus nun korrespondieren die Bewegungen des Cursor am Bildschirm mit denen des Gerätes auf der Tischfläche. Das Erreichen des gewünschten Feldes wird durch Drücken eines an der Maus befindlichen Knopfes dem Rechner signalisiert.

Soviel zu Anwendung und Entstehungsgeschichte der "Maus".

Um Ihnen die Möglichkeit zu geben, mit diesem reizvollen Tierchen eigene Versuche anzustellen ohne die Anschaffung eines kostspieligen Spinball, haben wir das folgende kleine Programm entwickelt, das mit einem gewöhnlichen Joystick am Controlport 2 Ihres C-64 arbeitet.

Natürlich läßt sich damit nicht derselbe Komfort erreichen wie mit der echten Maus, aber zur Erprobung des Prinzips ist es äußerst zweckdienlich.

Die Anwendung dieses Programms ist denkbar einfach. Nach RUN erscheint in der linken oberen Ecke des Bildschirmes der Cursor. Diesen können Sie nun mittels des Joystick, den Sie hoffentlich in Controlport 2 gesteckt haben, bewegen. Deckt er sich nun mit einem Zeichen, das Sie gerne nach B$ übernehmen möchten, so drücken Sie einfach den Knopf.

Zur Quittung erscheint das ausgewählte Zeichen einige Zeilen darunter. Nun suchen Sie das nächste Zeichen aus.

Nach Anwahl von RETURN ist die Zeile komplett in B$, aus dem Sie die Daten jetzt entnehmen können. Anschließend geht es auf die gleiche Weise mit der nächsten Zeile weiter.

Abschließend nun das Programmlisting:

```
1 dr=peek(56322):poke56322,224:ro$=chr$(18):rf$=chr$(146)
5 printchr$(147):gosub10:goto60!#=64!Ç*0=0:2=2
10 printchr$(19)" , - . / 0 1 2 3 4 5 6 7 8 9          ";
20 print"  # a b c d e f g h i j k l            ";
30 print" m n o p q r s t u v w x y z            ";
40 print" "ro$"ret"rf$" "ro$"del"rf$" "ro$"f1"rf$" "
   ro$"f3"rf$" "ro$"f5"rf$;!#=35!
45 print" "ro$"f7"rf$"                    "
50 return
60 dima(4,40)
100 fori=0to13
120 a(0,i*2+1)=i+44
140 nexti
180 fori=0to12
200 a(1,i*2+2)=i+64
220 nexti
260 fori=0to13
280 a(2,i*2+1)=i+77
300 nexti
340 fori=1to3
360 a(3,i)=13
380 nexti
```

```
420 fori=5to7
440 a(3,i)=20
460 nexti
500 fori=0to3
520 a(3,i*2+9)=i+133
540 a(3,i*2+9)=i+133
560 nexti
580 print:print
600 b$="":x=fre(0)
640 gosub5000:rem zeichen holen
680 printa$;
700 sys58643:pz=peek(211):ps=peek(214)
720 gosub10
760 poke211,pz:poke214,ps:remsys58636
780 ifasc(a$)=13then600
800 b$=b$+a$
820 goto640
5000 rem
5001 rem ***** joystick-abfrage *****
5002 rem
5020 sys58643:rem printcursor retten
5060 pz=peek(781):ps=peek(782)
5070 poke205,3
5080 ifpeek(207)then5080
5090 poke204,1
5100 poke781,0:poke782,0:jz=0:js=0
5120 sys58636:rem joystickcursor setzen
5140 poke204,0:rem cursor einschalten
5160 j=peek(56320):rem joystick abfragen
5170 fori=0to50:nexti
5180 if(j and1)=0thenjz=jz-1
5200 if(j and2)=0thenjz=jz+1
5220 if(j and4)=0thenjs=js-1
5240 if(j and8)=0thenjs=js+1
5260 if(j and16)=0then6000
5280 ifjz<0thenjz=0
5281 ifjs<0thenjs=0
5282 ifjs>30thenjs=30
5283 ifjz>3thenjz=3
5285 poke205,3
```

```
5290 ifpeek(207)then5290
5295 poke204,1
5300 poke781,jz:poke782,js:sys58636
5340 goto5140
6000 rem
6001 rem ***** zuordnung zu einem zeichen *****
6002 rem
6010 poke205,3
6015 ifpeek(207)then6015
6017 poke204,1
6020 sys58643:rem cursorposition holen
6040 z=peek(781):s=peek(782)
6050 ifs>39thens=s-40
6060 a$=chr$(a(z,s))
6100 poke781,pz:poke782,ps
6120 sys58636:rem printposition laden
6160 return
```

4.2 Der Joystick als komfortable Cursorsteuerung

Schön wäre es nun auch, zum Editieren von Programmen eine Art Maus-Simulation zu haben. Eine solche, mitunter auch sehr zeitaufwendige Routine, könnte natürlich nur in Assembler formuliert werden. Dabei würde sich dann folgendes Prinzip anbieten:

Zunächst "verbiegen" wir den Interruptzeiger auf unsere eigene Routine und springen dann zurück zum BASIC. Danach wird regelmäßig alle 1/60 Sekunde unsere Routine durchlaufen, wo wir dann die augenblickliche Joystickstellung herauslesen und entsprechend dazu dann den Cursor setzen könnten. Hier nun das Assemblerprogramm, kurz dahinter dann wieder der BASIC-Lader:

```
110:  cc00                    .opt p,oo
120:  cc00                    *=   $cc00
130:                  ;
140:                  ;*********************************
150:                  ;*** joystick-cursorsteuerung ****
160:                  ;*********************************
170:                  ;
180:  cc00 a9 0d              lda  #<irq   ; interrupt verbiegen
190:  cc02 a0 cc              ldy  #>irq
200:  cc04 78                 sei
210:  cc05 8d 14 03           sta  $314
220:  cc08 8c 15 03           sty  $315
230:  cc0b 58                 cli
240:  cc0c 60                 rts
250:                  ;
260:                  ;*** neue interuptroutine ***
270:                  ;
280:  cc0d ce 3f cc irq        dec  count   ; counter erniedrigen
290:  cc10 d0 12              bne  endirq  ; <> 0, dann fertig
300:  cc12 a9 04              lda  #4      ; counter neu laden
310:  cc14 8d 3f cc           sta  count
320:                  ;
330:  cc17 a2 05              ldx  #5      ; 5 joystickstellungen
340:  cc19 ad 00 dc           lda  56320   ; joystickport auslesen
350:  cc1c dd 27 cc loop      cmp  tab1,x  ; mit werten vergleichen
360:  cc1f f0 12              beq  found   ; gefunden !
370:  cc21 ca                 dex          ; sonst x=x-1
380:  cc22 10 f8              bpl  loop
390:  cc24 4c 31 ea endirq    jmp  $ea31
400:                  ;
410:  cc27 7e 77 7d tab1      .byt 126,119,125,123,111,110
420:  cc2d 91 1d 11 tab2      .byt 145,29,17,157,19,147
430:                  ;
440:                  ;*** wert gefunden ***
450:                  ;
460:  cc33 bd 2d cc found     lda  tab2,x  ; korrepondierenden wert
                                           ; laden
470:  cc36 8d 77 02           sta  631     ; in den tastaturbuffer
480:  cc39 a9 01              lda  #1      ; anzahl gedrueckter tasten
```

```
490:   cc3b 85 c6            sta  198     ; abspeichern
500:   cc3d d0 e5            bne  endirq  ; unbedingter sprung
510:                ;
520:   cc3f 04       count   .byt 4       ; counter
```

Der BASIC-Lader:

```
100 rem ******************************
110 rem ***                          ***
120 rem *** joystickcursorsteuerung ***
130 rem *** ---------------------- ***
140 rem ***                          ***
150 rem ******************************
160 :
170 for x=52224 to 52287
180 :read a
190 :poke x,a
200 :p=p+a
210 next x
220 :
230 if p<> 7435 then print "fehler in datas !!!":stop
240 :
250 end
260 :
50000 data 169, 13,160,204,120,141, 20
50010 data   3,140, 21,  3, 88, 96,206
50020 data  63,204,208, 18,169,  4,141
50030 data  63,204,162,  5,173,  0,220
50040 data 221, 39,204,240, 18,202, 16
50050 data 248, 76, 49,234,126,119,125
50060 data 123,111,110,145, 29, 17,157
50070 data  19,147,189, 45,204,141,119
50080 data   2,169,  1,133,198,208,229
50090 data   4
```

4.3 GEM-Simulation auf dem C64

Was die großen Computer können, das will auch ein kleiner Commodore 64 können - und es geht.

Am Beispiel eines Pull-Down-Menüs wird gezeigt, daß sich Fenstertechnik auch auf dem Commodore 64 realisieren läßt. Fenster sind Bereiche des Bildschirms, die vorübergehend mit neuen Inhalten gefüllt werden, ohne daß der alte Inhalt verloren geht. Das läßt sich z.B. nutzen, um eine komfortable Menüstruktur in ein Basic-Programm einzubauen. Geschickt eingesetzt, kann von jedem Programmteil aus das Pull-Down-Menü aufgerufen werden, etwa um Parameter zu ändern, um vielleicht in andere Programmteile hineinzuspringen oder auch nur um die Bildschirmfarbe zu ändern. An diese Beispiele hält sich das vorliegende Programm, um einige Möglichkeiten zu demonstrieren. Hier ist die Fenstertechnik kombiniert mit einer Maus-Simulation per Joystick, wobei 'jede Ähnlichkeit mit anderen Computern' sehr wohl beabsichtigt ist.

Das Pull-Down-Menü ist so strukturiert, daß es sowohl als eigenständiges Programm funktioniert, andererseits sinnvoll erweitert und für eigene Zwecke eingesetzt werden kann. Aus einem Dummy-Hauptprogramm heraus läßt sich mitten im Programmablauf blitzschnell das Menü aufrufen, bearbeiten und wieder in das Hauptprogramm zurückspringen, wonach das unterbrochene Programm an gleicher Stelle und mit gleichem Bildschirminhalt fortfährt.

In unserem Beispiel hat das Dummy-Programm die wenig ehrenvolle Aufgabe, den Bildschirm sinnlos und zufällig mit Zeichen zu füllen. An Dummy's Stelle tritt bei einer ernsthaften Anwendung natürlich das vom Nutzer geschriebene Hauptprogramm.

Nun folgt eine ausführliche Beschreibung des Pull-Down-Menü-Programms, auf daß der geneigte Leser sich besser zurechtfinden möge.

Initialisierung

Hier werden häufig benutzte Speicheradressen bestimmten Variablen zugeordnet und ein Sprite-Pfeil definiert (Zeile 230-240). Die Bedeutung der wichtigen Variablen ist in Remarks im Programmlisting angedeutet. Da der Bildschirm nach der Menü-Anzeige wieder im ursprünglichen Zustand sein soll, muß er weggespeichert und wiedergeholt werden können. Hierzu dienen die aus Zeitgründen in Maschinensprache geschriebenen Routinen, deren BASIC-Lader ab Zeile 260 bzw. 330 zu finden sind und mit SYS 52992 bzw. SYS 53072 aufgerufen werden.
Eigene Initialisierungs-Statements sollten diesem Teil angefügt werden.

Dummy-Hauptprogramm

Bei der eigenen Anwendung sollte die kleine Zufalls-Routine durch ein beliebiges Hauptprogramm ersetzt werden. Wichtig: Immer dort, wo ein Zugriff auf das Menü möglich sein soll, muß ein Feuerknopf-Test eingebaut werden. Im Beispiel ist dies in Zeile 500 realisiert. Da das Pull-Down-Menü als Unterprogramm ausgeführt ist, setzt sich nach dessen Abarbeitung das Hauptprogramm an der gleichen Stelle fort, wo es verlassen wurde. Man kann das Menü daher von verschiedenen Stellen aus aufrufen lassen, sogar von aus dem Menü heraus angewählten Programmteilen ist ein Rücksprung in das Hauptprogramm möglich.

Hauptmenü

Wird das Pull-Down-Menü angewählt, speichert SYS 52992 in Zeile 850 den aktuellen Bildschirminhalt und es erscheint zunächst eine Kopfzeile mit den Hauptpunkten und ein kleiner weißer Pfeil. Zeile 860 gibt die Hauptmenü-Felder auf den Bildschirm aus, sie ist jeweils den eigenen Bedürfnissen gemäß zu ändern. Per Joystick fährt man nun den Pfeil auf das gewünschte Feld und betätigt den Feuerknopf. In den Zeilen 900 bis 940 wird die Pfeilposition abgefragt und entsprechend in die

Untermenüs 1 bis 5 verzweigt. 'Klickt' man dagegen eine Stelle außerhalb der Menü-Felder an, so restauriert sich der Bildschirm und das Programm kehrt brav an die Stelle zurück, die es verlassen hatte.

Drei der fünf möglichen Untermenuü enthalten sinnvolle Beispiele einer Anwendung, die jeweils unterschiedlich programmiert werden müssen: Im ersten wird die Verzweigung in andere Programmteile vorgeführt, im zweiten lassen sich bestimmte Programm-Parameter setzen und löschen, im dritten wird die ach so beliebte Bildschirmfarbänderung ermöglicht. Bei einer eigenen Anwendung können die vorgeführten Beispiele verändert und den speziellen Bedürfnissen angepaßt werden.

Untermenü 1: Verzweigung

Alle Untermenüs haben formal den gleichen Aufbau, der am Beispiel des Untermenüs 1 erläutert wird. Die Zeilen 1050 bis 1120 geben das Pull-Down-Untermenü 1 (sechs Felder) an der richtigen Stelle aus, dabei hat man 14 Zeichen pro Feld zu dessen Kennzeichnung. Nach der Pfeilpositionierung (Zeile 1140) wird dessen Standort abgefragt (in 'h' die Spalte, in 'g' die Zeile) und dann ggf. die Sprungadresse (9050 bis 9150) berechnet (Zeile 1150 und 1180). Befindet sich der Pfeil außerhalb der Felder, restauriert sich der Bildschirm (Zeile 1160). Wurde in der ersten Bildschirmzeile 'geklickt', wird ein weiteres Hauptmenü-Feld abgefragt (Verzweigung nach Zeile 860), ansonsten kehrt man (in 1170) zum Hauptprogramm zurück.

Untermenü 2: Parameter ändern

Der Pull-Down-Bildschirm-Aufbau und die Abfrage ist in diesem und den folgenden Untermenüs ähnlich dem des Untermenüs 1. Was sich jeweils ändert ist die Ausgabestelle auf dem Bildschirm (mittels Cursor-Positionierung, z.B. 's=2.05') sowie die Spaltenabfrage des Pfeils (in 'h', z.B. Zeile 2190). Die jeweiligen Werte entnimmt man den entsprechenden Programmstellen bzw. den Zeilen 4040 und 5040.

Das Untermenü 2 setzt (=1) und löscht (=0) in diesem Fall sechs Parameter, welche in der indizierten Variable 'pa(6)' zu finden sind und in anderen Programmteilen ausgewertet werden können. Die Bearbeitung erfolgt in Zeile 2220 je nach Pfeilposition. Der Zustand der Parameter wird in den Zeilen 2130 bis 2160 sichtbar gemacht: dazu werden die Pull-Down-Felder ausgelesen und invers dargestellt.

Untermenü 3: Farbwahl

In diesem Untermenü wird am Beispiel der Farbwahl das Hochzählen von Variablen demonstriert. Wesentliche Unterschiede zu Untermenü 1 sind in den Zeilen 3150 bis 3180 zu finden: hier bewirkt das 'anklicken' der Menüfelder den Wechsel der Rahmen- bzw. Hintergrundfarbe.

Da nur zwei Untermenü-Felder benötigt werden, ist hier die Pull-Down-Ausgabe auf dem Bildschirm gegenüber den Untermenüs 1 und 2 verkürzt (Zeilen 3050 bis 3080). Die Zeilenabfrage des Pfeils wird hierzu geändert (Zeile 3110: '4' statt wie in den Untermenüs 1 und 2 '8') und die Sprungliste in Zeile 3140 ist kürzer. Entsprechend könnte ein Pull-Down-Untermenü natürlich auch verlängert werden.

Untermenü 4 und 5:

Beide Untermenüs sind hier aus Platzgründen nicht aufgeführt, werden aber wie bisher strukturiert: Die für die Positionierung wichtigen Angaben findet man in den Zeilen 4040 und 5040 hinter 'rem'.

Cursor positionieren

Es ist vielleicht schon aufgefallen, daß die Zeichenfolge "s='z'.'sp':gosub6000:" im Programm häufig auftaucht. Dabei

handelt es sich um eine Möglichkeit, den Cursor ohne viele Steuerzeichen in die Zeile 'z' und Spalte 'sp' zu positionieren. Programmzeile 6090, als Unterprogramm ausgeführt, POKEt den Cursor an die angegebene Stelle.

Pfeil positionieren

Dieser Teil ist aus einem Maus-Simulationsprogramm fast unverändert übernommen (siehe *DATAWELT* 11/85, Seite 79/80).

Verzweigungsadressen

Die Zeilen 9050 bis 9150 sind in unserem Beispiel als Verzweigungsadressen des Untermenüs 1 ausgeführt: Der Pfeil wird abgeschaltet, der Bildschirm restauriert und das Programm endet. Hier könnten Programmteile beginnen, in die über das Pull-Down-Menü wahlweise eingesprungen wird.

4.4 Ausgabe deutscher Fehlermeldungen

Stört es Sie eigentlich, daß Ihr Commodore 64 englisch mit Ihnen 'spricht'? - Dann bringen Sie ihm doch Deutsch bei!

Wenn Sie das folgende Programm auf Ihrem 64er starten, wird Ihr Rechner dazu veranlaßt, sämtliche Fehlermeldungen in deutscher Sprache auszugeben. Im Programm steht eine Liste der Fehlermeldungen in deutscher Sprache. Die Fehlermeldungen sind von 1 bis 29 numeriert, die erste Fehlermeldung wäre z.B. 'too many files'. Diese Tabelle finden Sie in '64 intern'. Wir ersetzen einfach die Fehlermeldungen dieser Tabelle durch eigene deutsche Texte. Besitzen Sie einen Assembler für den 64er und wollen eventuell selbst andere Texte eingeben, so beachten Sie bitte, daß das Ende eines Textes jeweils durch ein Zeichen mit gesetztem Bit 7 gekennzeichnet ist. Sie können das einfach

dadurch erreichen, indem Sie den letzten Buchstaben eines jeden Textes geshiftet eingeben. Für die 'Nur-BASIC-Programmierer' haben wir wieder ein Ladeprogramm abgedruckt.

```
100:   c000                    .opt p1
;
; deutsche fehlermeldungen c64
;
140:   c000          errvec    =    $300    ; vector für fehlermeldungen
150:   c000          erradr    =    $22     ; zeiger auf fehlermeldungen
160:   c000          olderr    =    $a445
;
180:   c000                    *=   $c000
190:   c000 a9 0b    init      lda  #<newerr
200:   c002 a0 c0              ldy  #>newerr
210:   c004 8d 00 03           sta  errvec
220:   c007 8c 01 03           sty  errvec+1
230:   c00a 60                 rts
;
250:   c00b 8a       newerr    txa            ; fehlernummer
260:   c00c 10 03              bpl  err1
270:   c00e 4c 43 c0           jmp  ready
280:   c011 0a       err1      asl  a         ; mal 2
290:   c012 aa                 tax            ; als index
300:   c013 bd 48 c0           lda  errtab-2,x
310:   c016 85 22              sta  erradr
320:   c018 bd 49 c0           lda  errtab-1,x
330:   c01b 85 23              sta  erradr+1
340:   c01d 20 cc ff           jsr  $ffcc     ; clrch
350:   c020 a9 00              lda  #0
360:   c022 85 13              sta  $13       ; i/o-flag
370:   c024 20 d7 aa           jsr  $aad7     ; neue zeile
380:   c027 20 45 ab           jsr  $ab45     ; "?" ausgeben
390:   c02a a0 00              ldy  #0
400:   c02c b1 22    loop      lda  (erradr),y
400:   c02e 48                 pha
400:   c02f 29 7f              and  #$7f
410:   c031 20 47 ab           jsr  $ab47
420:   c034 c8                 iny
```

```
420:  c035 68              pla
420:  c036 10 f4           bpl  loop
430:  c038 20 7a a6        jsr  $a67a   ; cont sperren
440:  c03b a4 3a           ldy  $3a
440:  c03d c8              iny
440:  c03e f0 03           beq  ready
450:  c040 20 c2 bd        jsr  $bdc2   ; in zeilennummer
460:  c043 a9 bb    ready  lda  #< fertig
470:  c045 a0 c2           ldy  #> fertig
480:  c047 20 78 a4        jsr  $a478   ; wie gehabt
490:  c04a 86 c0 96 errtab .word f1,f2,f3,f4,f5,f6,f7,f8
500:  c05a 13 c1 2b        .word f9,f10,f11,f12,f13,f14,f15
510:  c068 89 c1 9f        .word f16,f17,f18,f19,f20,f21,f22
520:  c076 15 c2 29        .word f23,f24,f25,f26,f27,f28,f29,f30
;
540:  c086 5a 55 20 f1     .asc "zu viele dateieN"
550:  c096 44 41 54 f2     .asc "datei offeN"
560:  c0a1 44 41 54 f3     .asc "datei nicht offeN"
570:  c0b2 44 41 54 f4     .asc "datei nicht gefundeN"
580:  c0c6 47 45 52 f5     .asc "geraet nicht vorhandeN"
590:  c0dc 4b 45 49 f6     .asc "keine eingabedateI"
600:  c0ee 4b 45 49 f7     .asc "keine ausgabedateI"
610:  c100 46 45 48 f8     .asc "fehlender dateinamE"
620:  c113 55 4e 47 f9     .asc "ungueltige geraetenummeR"
630:  c12b 27 4e 45 f10    .asc "'next' ohne 'for"
640:  c13b a7              .byt "'"+$80
650:  c13c 53 59 4e f11    .asc "syntax fehleR"
660:  c149 27 52 45 f12    .asc "'return' ohne 'gosub"
670:  c15d a7              .byt "'"+$80
680:  c15e 5a 55 20 f13    .asc "zu wenig 'datas"
690:  c16d a7              .byt "'"+$80
700:  c16e 55 4e 5a f14    .asc "unzulaessiger werT"
710:  c180 55 45 42 f15    .asc "ueberlauF"
720:  c189 5a 55 20 f16    .asc "zu wenig speicherplatZ"
730:  c19f 46 45 48 f17    .asc "fehlende zeilE"
740:  c1ad 55 4e 47 f18    .asc "ungueltiger indeX"
750:  c1be 46 45 4c f19    .asc "feld bereits dimensionierT"
760:  c1d8 44 49 56 f20    .asc "division durch nulL"
770:  c1eb 44 49 52 f21    .asc "direkt-modus verboteN"
780:  c200 46 41 4c f22    .asc "falscher variablentyP"
```

```
790:   c215 5a 45 49 f23      .asc "zeichenkette zu lanG"
800:   c229 46 45 48 f24      .asc "fehlerhafte daten aus dateI"
810:   c244 53 54 52 f25      .asc "stringausdruck zu komplizierT"
820:   c261 46 4f 52 f26      .asc "fortfuehren nicht moeglicH"
830:   c27b 4e 49 43 f27      .asc "nicht definierte funktioN"
840:   c294 4b 45 49 f28      .asc "keine uebereinstimmunG"
850:   c2aa 4c 41 44 f29      .asc "ladefehleR"
860:   c2b4 41 42 42 f30      .asc "abbrucH"
870:   c2bb 0d       fertig   .byt 13      ; crlf
880:   c2bc 46 45 52          .asc "fertig."
890:   c2c3 0d 00             .byt 13,0
```

Der Basiclader:

```
100 for i = 49152 to 49860
110 read x : poke i,x : s=s+x : next
120 data 169, 11,160,192,141,  0,  3,140,  1,  3, 96,138
130 data  16,  3, 76, 67,192, 10,170,189, 72,192,133, 34
140 data 189, 73,192,133, 35, 32,204,255,169,  0,133, 19
150 data  32,215,170, 32, 69,171,160,  0,177, 34, 72, 41
160 data 127, 32, 71,171,200,104, 16,244, 32,122,166,164
170 data  58,200,240,  3, 32,194,189,169,187,160,194, 32
180 data 120,164,134,192,150,192,161,192,178,192,198,192
190 data 220,192,238,192,  0,193, 19,193, 43,193, 60,193
200 data  73,193, 94,193,110,193,128,193,137,193,159,193
210 data 173,193,190,193,216,193,235,193,  0,194, 21,194
220 data  41,194, 68,194, 97,194,123,194,148,194,170,194
230 data 180,194, 90, 85, 32, 86, 73, 69, 76, 69, 32, 68
240 data  65, 84, 69, 73, 69,206, 68, 65, 84, 69, 73, 32
250 data  79, 70, 70, 69,206, 68, 65, 84, 69, 73, 32, 78
260 data  73, 67, 72, 84, 32, 79, 70, 70, 69,206, 68, 65
270 data  84, 69, 73, 32, 78, 73, 67, 72, 84, 32, 71, 69
280 data  70, 85, 78, 68, 69,206, 71, 69, 82, 65, 69, 84
290 data  32, 78, 73, 67, 72, 84, 32, 86, 79, 82, 72, 65
300 data  78, 68, 69,206, 75, 69, 73, 78, 69, 32, 69, 73
310 data  78, 71, 65, 66, 69, 68, 65, 84, 69,201, 75, 69
320 data  73, 78, 69, 32, 65, 85, 83, 71, 65, 66, 69, 68
330 data  65, 84, 69,201, 70, 69, 72, 76, 69, 78, 68, 69
```

```
340 data  82, 32, 68, 65, 84, 69, 73, 78, 65, 77,197, 85
350 data  78, 71, 85, 69, 76, 84, 73, 71, 69, 32, 71, 69
360 data  82, 65, 69, 84, 69, 78, 85, 77, 77, 69,210, 39
370 data  78, 69, 88, 84, 39, 32, 79, 72, 78, 69, 32, 39
380 data  70, 79, 82,167, 83, 89, 78, 84, 65, 88, 32, 70
390 data  69, 72, 76, 69,210, 39, 82, 69, 84, 85, 82, 78
400 data  39, 32, 79, 72, 78, 69, 32, 39, 71, 79, 83, 85
410 data  66,167, 90, 85, 32, 87, 69, 78, 73, 71, 32, 39
420 data  68, 65, 84, 65, 83,167, 85, 78, 90, 85, 76, 65
430 data  69, 83, 83, 73, 71, 69, 82, 32, 87, 69, 82,212
440 data  85, 69, 66, 69, 82, 76, 65, 85,198, 90, 85, 32
450 data  87, 69, 78, 73, 71, 32, 83, 80, 69, 73, 67, 72
460 data  69, 82, 80, 76, 65, 84,218, 70, 69, 72, 76, 69
470 data  78, 68, 69, 32, 90, 69, 73, 76,197, 85, 78, 71
480 data  85, 69, 76, 84, 73, 71, 69, 82, 32, 73, 78, 68
490 data  69,216, 70, 69, 76, 68, 32, 66, 69, 82, 69, 73
500 data  84, 83, 32, 68, 73, 77, 69, 78, 83, 73, 79, 78
510 data  73, 69, 82,212, 68, 73, 86, 73, 83, 73, 79, 78
520 data  32, 68, 85, 82, 67, 72, 32, 78, 85, 76,204, 68
530 data  73, 82, 69, 75, 84, 45, 77, 79, 68, 85, 83, 32
540 data  86, 69, 82, 66, 79, 84, 69,206, 70, 65, 76, 83
550 data  67, 72, 69, 82, 32, 86, 65, 82, 73, 65, 66, 76
560 data  69, 78, 84, 89,208, 90, 69, 73, 67, 72, 69, 78
570 data  75, 69, 84, 84, 69, 32, 90, 85, 32, 76, 65, 78
580 data 199, 70, 69, 72, 76, 69, 82, 72, 65, 70, 84, 69
590 data  32, 68, 65, 84, 69, 78, 32, 65, 85, 83, 32, 68
600 data  65, 84, 69,201, 83, 84, 82, 73, 78, 71, 65, 85
610 data  83, 68, 82, 85, 67, 75, 32, 90, 85, 32, 75, 79
620 data  77, 80, 76, 73, 90, 73, 69, 82,212, 70, 79, 82
630 data  84, 70, 85, 69, 72, 82, 69, 78, 32, 78, 73, 67
640 data  72, 84, 32, 77, 79, 69, 71, 76, 73, 67,200, 78
650 data  73, 67, 72, 84, 32, 68, 69, 70, 73, 78, 73, 69
660 data  82, 84, 69, 32, 70, 85, 78, 75, 84, 73, 79,206
670 data  75, 69, 73, 78, 69, 32, 85, 69, 66, 69, 82, 69
680 data  73, 78, 83, 84, 73, 77, 77, 85, 78,199, 76, 65
690 data  68, 69, 70, 69, 72, 76, 69,210, 65, 66, 66, 82
700 data  85, 67,200, 13, 70, 69, 82, 84, 73, 71, 46, 13
710 data   0
720 if s <> 62278 then print "fehler in datas !!" : end
730 sys 49152 : print "ok !"
```

4.5 Die Tastaturbelegung und ihre Änderung

Die Tastatur des Commodore 64 ist als eine Matrix mit 8 Zeilen und 8 Spalten angeordnet. Dabei sind die Leitungen der 8 Zeilen mit Port A (Adresse $DC00 = 56320) von CIA 1 verbunden und die 8 Spalten liegen an Port B (Adresse $DC01 = 56321) von CIA 1. Bei der Tastaturabfrage (Adresse $FF9F = 65485) wird die Tastatur zeilenweise abgefragt, indem nacheinander auf jede Zeile über den Port A ein Signal angelegt wird. Ist nun eine Taste gedrückt, so kann man über Port B die Spalte der gerade gedrückten Taste abfragen. Aus Zeilen- und Spaltennummer wird nun die Tastennummer zwischen 0 und 63 berechnet. 64 bedeutet hier keine Taste gedrückt. Die Zuordnung ist in der folgende Tabelle dargestellt. Dieser Wert wird bei jeder Abfrage in Adresse $CB gleich 203 abgelegt; die Nummer der letzten gedrückten Taste wird in $C5 = 197 hinterlegt. Der Status der Sondertasten wird bei der Tastaturabfrage in Adresse $028D = 653 gespeichert. Bit 0 bedeutet SHIFT, Bit 1 ist für COMMODORE reserviert und Bit 2 steht für gedrückte CTRL-Taste. Die Zuordnung eines bestimmten Zeichens zu einer Taste geschieht über verschiedene Tabellen, die den ASCII-Code bestimmen, der einer bestimmten Taste zugeordnet wird. Da im Commodre 64 alle Tasten vierfach belegt sein können, existieren 4 solcher Tabellen. Beachten Sie auch die Unterscheidung zwischen linker und rechter Shift-Taste. Shift Lock ist mit der linken Shift-Taste verbunden.

Spalte	0	1	2	3	4	5	6	7
Zeile								
0	DEL	RETURN	CURRIGHT	F7	F1	F3	F5	CURDOWN
1	3	W	A	4	Z	S	E	SHIFT LINKS
2	5	R	D	6	C	F	T	X
3	7	Y	G	8	B	H	U	V
4	9	I	J	0	M	K	O	N
5	+	P	L	-	.	:	#	,
6	PFUND	*	;	HOME	SHIFT RECHTS	=	^	/
7	1	PFL	CTRL	2	LEER	C=	Q	STOP

Die erste Tabelle gibt den ASCII-Code, wenn die Taste allein gedrückt wurde. In der zweiten Tabelle steht der Code mit gedrückter Shift-Taste, Tabelle 3 gilt für gedrückte Commodore-Taste und Tabelle 4 schließlich ist für die Control-Taste zuständig. Ein Eintrag von $FF = 255 in dieser Tabelle markiert einen ungültigen Eintrag. Die Tasten 'Shift', 'Commodore' und 'CTRL' werden gesondert behandelt; die entsprechenden Einträge in der ersten Tabelle sind 1, 2 und 4. Dieser Status wird zusätzlich in $28D = 653 gespeichert. Die Bits 0, 1 und 2 entsprechen diesen drei Tasten.

Wollen wir nun einer Taste einen anderen Code zuordnen, müssen wir den entsprechenden Eintrag in der Tabelle ändern. Da die Tabelle im ROM liegt, ist dies nicht ohne weiteres möglich. Der Commodore 64 verfügt jedoch auch über RAM in diesem Speicherbereich, so daß wir das Betriebssystem in das 'darunter' liegende RAM kopieren und dort die Änderung vornehmen können. Dies kann mit einem kleinen BASIC-Programm geschehen. Gleichzeitig muß auch das BASIC ins darunterliegende RAM kopiert werden.

```
100 for i = 40960 to 49151: rem basic-rom kopieren
110 poke i, peek(i) : next
120 for i = 14*4096 to 65535 : rem betriebssystem kopieren
130 poke i, peek(i) : next
140 poke 1, 53
```

Die Zeilen 100 bis 130 kopieren das Betriebssystem und BASIC vom ROM in das darunter liegende RAM. In Zeile 140 wird dann vom ROM auf RAM umgeschaltet, so daß das Betriebssystem jetzt im RAM läuft. Jetzt können wir hingehen und die Codes einzelner Tasten ändern.

Bei der deutschen Tastatur sind normalerweise Y und Z gegenüber der amerikanischen Tastatur vertauscht, d.h. Z liegt neben T und Y liegt neben X.

Dazu brauchen wir zuerst die Adressen der 4 Tabellen:

```
Tabelle 1  ungeshifted    $EB81 = 60289
Tabelle 2  mit shift      $EBC2 = 60354
Tabelle 3  mit Commodore  $EC03 = 60419
Tabelle 4  mit Control    $EC78 = 60536
```

Wollen wir jetzt einen Code ändern, so müssen wir aus der Matrixtabelle die Nummer der Tasten ermitteln, die wir ändern wollen. Die Numerierung läuft von 0 links oben bis 63 rechts unten. Y hat also die Nummer 25 und Z die Nummer 12. In diese Adressen müssen wir also nun den neuen Code schreiben.

```
150 t1 = 60289  : rem tabelle 1 ohne shift
160 t2 = 60354  : rem tabelle 2 mit shift
170 t3 = 60419  : rem tabelle 3 mit commodore
180 t4 = 60536  : rem tabelle 4 mit control
200 poke t1+25, asc("z")
210 poke t1+12, asc("y")
```

Drücken Sie jetzt die Taste "Z", so erscheint ein "y" auf dem Bildschirm und umgekehrt. Um Klein- und Großbuchstaben zu ändern, müssen Sie die Einträge in Tabelle 2 ebenfalls ändern:

```
220 poke t2+25, asc("Z")
230 poke t2+12, asc("Y")
```

Werden die entsprechenden Control-Tasten benötigt, so können Sie auch diese ändern:

```
240 poke t4+25, asc("z")-64
250 poke t4+12, asc("y")-64
```

Mit Hilfe dieser 4 Tabellen können Sie also 4*64 gleich 256 verschiedene Zeichen definieren. Die RESTORE-Taste können Sie nicht umdefinieren; sie ist direkt mit der NMI-Leitung des Prozessors verbunden. Die Tastaturdefinition bleibt solange erhalten, bis Sie RUN/STOP-Restore drücken. Dabei wird wieder auf das ROM zurückgeschaltet. Aber auch dies können Sie verhindern, indem Sie den Wert für die Speicherkonfiguration im RAM ändern. Dies könnte in unserem obigen Programm z.B. in Zeile 190 geschehen:

```
190 poke 64982, 53
```

Anstatt die Nummer der Taste aus der Matrix zu ermitteln, kann man sie auch vom Programm selbst bestimmen lassen. Ein Programm mit automatischer Ermittlung der Nummer könnte z.B. so aussehen:

```
100 dim t(4): for i=1 to 4: read t(i): next
110 data 60289,60354,60419,60536
120 for i=14*4096 to 65535: poke i, peek(i): next
130 for i=40960 to 49151 : poke i, peek(i) : next
140 poke1, 53: poke64982, 53
1000 print "bitte druecken sie die zu aendernde taste ";
1010 get a$: if a$="" then1010
1020 print a$
1030 a = asc(a$)
1040 for j=1 to 4: t=t(j)
1050 for i=0 to 63: if peek(t+i) <> a then next: next
```

```
1060 print "bitte gewuenschte belegung angeben ";
1070 get a$: if a$="" then 1070
1080 print a$
1090 poke t+i, asc(a$): goto 1000
```

4.6 Die Belegung der Funktionstasten

Der Commodore 64 hat neben seinen alphanumerischen Tasten noch 4 Funktionstasten, die doppelt belegt sind. Diese Tasten kann man z.B. bei der Menüsteuerung benutzen, um bestimmte Programmteile auszuwählen. Dazu kann man mit GET die Tastatur abfragen und abhängig von der gedrückten Taste die Verzweigung vornehmen. Die Funktionstasten haben folgende ASCII-Codes

```
F1  =>  133
F3  =>  134
F5  =>  135
F7  =>  136
```

Bei gleichzeitig gedrückter SHIFT-Taste erhöht sich der ASCII-Code um 4:

```
F2  =>  137
F4  =>  138
F6  =>  139
F8  =>  140
```

Eine Funktionstastenabfrage kann man z.B. so verwirklichen:

```
100 GET A$ : IF A$="" THEN 100
110 A = ASC (A$)
120 IF A = 133 THEN 1100 : REM F1
130 IF A = 134 THEN 1200 : REM F3
```

```
140 IF A = 135 THEN 1300 : REM F5
150 IF A = 136 THEN 1400 : REM F7
160 IF A = 137 THEN 1500 : REM F2
170 IF A = 138 THEN 1600 : REM F4
180 IF A = 139 THEN 1700 : REM F6
190 IF A = 140 THEN 1800 : REM F8
200 GOTO 100
```

Abhängig von der gedrückten Funktionstaste wird in die entsprechende Routine verzweigt. Eleganter kann man dies mit einem ON .. GOTO Sprungverteiler lösen.

```
100 GET A$ : IF A$="" THEN 100
110 A = ASC (A$) : IF A<133 OR A>140 THEN 100
120 ON A-132 GOTO 1100,1200,1300,1400,1500,1600,1700,1800
```

Diese Technik können Sie innerhalb von Programmen benutzen. Wir wollen Ihnen jetzt jedoch ein Programm vorstellen, daß es erlaubt, jeder Funktionstaste einen Text zuzuweisen, der beim Drücken der Taste auf dem Bildschirm erscheint. Damit können Sie sich die Arbeit mit dem Rechner komfortabler gestalten. Meist wird man die Funktionstasten mit BASIC-Befehlsworten belegen, z.B. RUN, LIST, PRINT, GOTO. Ebenso wird es möglich sein, einer Funktionstaste ein Wort einschließlich der abschließenden RETURN-Taste zuzuweisen. Damit werden die obigen Befehle dann direkt ausgeführt. Steht z.B. auf der Taste F1 der Text LIST und der Code für RETURN, so wird durch Drücken der Taste F1 das Programm gelistet, ohne daß eine weitere Tastenbetätigung erforderlich wäre. Die maximale Länge eines Textes bei der Belegung der Funktionstasten ist 10 Zeichen, die Länge des Tastatureingabepuffers.

Als Besonderheit können Sie mit unserem Programm nicht nur 8 verschiedene Texte auf die Funktionstasten legen (doppelte Belegung - mit und ohne SHIFT-Taste), sondern sogar 16. Zur

Unterscheidung dienen dabei außer der SHIFT-Taste noch die Commodore- und die Control-Taste. Wir wählen folgende Zuordnung zu den Tasten:

```
F1  =>   F1
F2  =>   F3
F3  =>   F5
F4  =>   F7
F5  =>   F1  plus SHIFT
F6  =>   F3  plus SHIFT
F7  =>   F5  plus SHIFT
F8  =>   F7  plus SHIFT
F9  =>   F1  plus COMMODORE
F10 =>   F3  plus COMMODORE
F11 =>   F5  plus COMMODORE
F12 =>   F7  plus COMMODORE
F13 =>   F1  plus CTRL
F14 =>   F3  plus CTRL
F15 =>   F5  plus CTRL
F16 =>   F7  plus CTRL
```

Wir stellen Ihnen nun ein Maschinenprogramm vor, das die Belegung der Funktionstasten mit den beliebigen Texten gestattet. Diese Texte sind als DATA's abgelegt und können bis zu 10 Zeichen lang sein.

```
; Funktionstasten für cbm 64
;
190:  033c                *=   828     ; Kassettenpuffer
200:  033c        keyvec  =    $28f    ; Vector für
                                       ; Tastaturdekodierung
210:  033c        keypnt  =    $f5     ; Zeiger auf Dekodiertabelle
220:  033c        buffer  =    $277    ; Tastaturpuffer
230:  033c        nmbkeys =    $c6     ; Anzahl der Zeichen im
                                       ; Tastaturpuffer
240:  033c        shift   =    $28d    ; Flag für SHIFT/ CBM/ CTRL
250:  033c        keyno   =    $cb     ; Matrixnummer der gedrückten
                                       ; Taste
```

```
260:  033c           lastkey  =   $c5     ; Nummer der letzten Taste
265:  033c           temp     =   lastkey
270:  033c           fmin     =   $85     ; Code der niedrigsten
                                          ; Funktionstaste
280:  033c           fmax     =   $88     ; Code der höchsten
                                          ; Funktionstaste
285:  033c           setflag  =   $eb26
287:  033c           oldkey   =   $eb48   ; alte Tastaturauswertung
;
300:  033c a9 47     init     lda #<funktion  ; Vector auf neue Routine
                                          ; setzen
300:  033e a0 03              ldy #>funktion
310:  0340 8d 8f 02           sta keyvec
310:  0343 8c 90 02           sty keyvec+1
310:  0346 60                 rts
;
330:  0347 a4 cb     funktion ldy keyno   ; Nummer der Taste
335:  0349 c4 c5              cpy lastkey ; gleiche wie vorher ?
335:  034b f0 0a              beq nofunc  ; ja
340:  034d b1 f5              lda (keypnt),y  ; ASCII-Code
350:  034f c9 89              cmp #fmax+1   ; mit höchster
                                          ; Funktionstaste vergleichen
360:  0351 b0 04              bcs nofunc  ; keine Funktionstaste ?
370:  0353 c9 85              cmp #fmin
380:  0355 b0 03              bcs function
390:  0357 4c 48 eb nofunc    jmp oldkey  ; zur alten
                                          ; Tastaturauswertung
400:  035a e9 85     function sbc #fmin
400:  035c 85 c5              sta temp
410:  035e 0a                 asl
410:  035f 0a                 asl         ; mal 10
410:  0360 65 c5              adc temp
410:  0362 0a                 asl
420:  0363 ae 8d 02           ldx shift   ; Flag SHIFT/CBM CTRL
430:  0366 e0 01              cpx #1      ; SHIFT ?
430:  0368 f0 0e              beq shiftkey
440:  036a e0 02              cpx #2      ; CBM ?
440:  036c f0 07              beq cbmkey
450:  036e e0 04              cpx #4      ; CTRL ?
```

```
450:  0370 d0 09          bne  nospec
500:  0372 18      ctrlkey clc
500:  0373 69 28          adc  #40
510:  0375 18      cbmkey  clc
510:  0376 69 28          adc  #40      ; Zeiger auf nächste Tabelle
520:  0378 18      shiftkey clc
520:  0379 69 28          adc  #40
530:  037b aa      nospec  tax           ; Zeiger als Index
540:  037c a0 00          ldy  #0
550:  037e bd 00 cf getkey  lda  tabel,x ; Belegung aus Tabelle holen
550:  0381 f0 09          beq  endkey
560:  0383 99 77 02       sta  buffer,y ; und in Tastaturpuffer
                                        ; schreiben
570:  0386 e8             inx
570:  0387 c8             iny
570:  0388 c0 0a          cpy  #10      ; schon 10 Zeichen ?
580:  038a d0 f2          bne  getkey
590:  038c 84 c6   endkey sty  nmbkeys  ; Anzahl der Zeichen merken
600:  038e a2 ff          ldx  #$ff     ; Flag für Tastaturcode
                                        ; ungültig
610:  0390 4c 26 eb       jmp  setflag  ; Flags aktualisieren
615:  0393         tabel  =    $cf00    ; Tabelle mit
                                        ; Tastenbelegungen
```

Das folgende BASIC-Programm generiert das Maschinenprogramm und legt die Texte für die 16 Funktionstasten im Speicher ab. Die Texte selbst sind als DATA's ab Zeile 300 im Programm hinterlegt und können natürlich nach Ihren Wünschen geändert werden. Beachten Sie dabei jedoch, daß Texte mit mehr als 10 Zeichen nicht zulässig sind. Wollen Sie, daß ein Befehl durch Drücken der Funktionstaste direkt ausgeführt wird, so muß das abschließende 'RETURN' mit im Text enthalten sein. Schreiben Sie dazu als letztes Zeichen einen Pfeil nach links. Dieses Zeichen wird beim Laden dann gegen den RETURN-Code ausgetauscht (Zeile 250). Wollen Sie Anführungszeichen innerhalb des Textes benutzen, so verwenden Sie statt dessen ein Apostroph ': LOAD'$',8 (Zeile 260). Kommt innerhalb eines Textes ein Komma vor, so muß der komplette Text in Hochkommas eingeschlossen werden (siehe Zeile 330).

```
100 for i = 828 to 914
110 read x : poke i,x : s=s+x : next
120 data 169, 71,160,  3,141,143,  2,140,144,  2, 96,164
130 data 203,196,197,240, 10,177,245,201,137,176,  4,201
140 data 133,176,  3, 76, 72,235,233,133,133,197, 10, 10
150 data 101,197, 10,174,141,  2,224,  1,240, 14,224,  2
160 data 240,  7,224,  4,208,  9, 24,105, 40, 24,105, 40
170 data  24,105, 40,170,160,  0,189,  0,207,240,  9,153
180 data 119,  2,232,200,192, 10,208,242,132,198,162,255
190 data  76, 38,235
200 if s <> 10591 then print "fehler in datas !!" : end
210 sys 828
220 rem belegungen im speicher ablegen
230 ad = 12*4096+15*256 : rem $cf00
240 for i=0 to 15 : read x$ : for j=1 to len(x$)
250 a=asc(mid$(x$,j,1)) : if a=95 then a=13 :rem return
260 if a= 39 then a=34 : rem anführungszeichen
270 poke ad+10*i+j-1,a : next
280 if j<>11 then poke ad+i*10+j-1,0 : rem endekriterium
290 next
300 data list, run, goto, chr$(
310 data ?fre(0), save, print, then
320 data poke, peek(, print#, input#
330 data "load'$'$,8", next, gosub, return
```

Die Texte der Funktionstasten werden im freien RAM-Bereich ab Adresse $CF00 abgelegt. Ist dieser Bereich bei Ihnen belegt oder wollen Sie einen anderen Speicherbereich für die Tastentexte benutzen, müssen Sie diese Adresse in Zeile 230 einsetzen und zusätzlich die Adresse in den DATA-Statements in Zeile 170 ändern. Ersetzen Sie dazu das fünft- und viertletzte Element (0 und 207) durch Low- und High-Byte der neuen Adresse. Wollen Sie eine andere Adresse benutzen, so müssen dort 160 Bytes zur Verfügung stehen (16 Tasten mal 10 Zeichen).

Wollen Sie die Funktionstastenbelegung ändern, so genügt es, das Programm mit entsprechend geänderten Datas ab Zeile 220 laufen zu lassen.

Durch Drücken von RUN/STOP-RESTORE wird die Funktionstastendefinition wieder aufgehoben. Mit

```
SYS 828
```

können Sie in einem solchen Falle die Belegungen wieder aktivieren.

5. Kapitel

Peripherie

5.1 Ein kleiner Kopierschutz

Jeder, der einmal ein Programm geschrieben hat, weiß, was er da geleistet hat. Stunden, ja Tage, Wochen oder Monate voller Mühe stecken in einem guten Programm. Wer will sich da nicht einigermaßen gegen eine unerlaubte Vervielfältigung seiner Arbeit und seiner Idee absichern? Das ist allerdings leichter gesagt als getan, denn auf der ganzen Welt gibt es eine Gemeinde von Computerfreaks, die es sich zur Aufgabe gemacht hat, jeden neuen Kopierschutz und sei er noch so perfekt, zu knacken. Das ist teilweise zu einem richtigen Volkssport geworden.

Ferner gibt es heute immer ausgefeiltere Kopierprogramme, mit denen es auch dem blutigen Laien möglich ist, geschützte Programme zu kopieren, da die vorhandene Sicherung des Programms einfach von dem Kopierprogramm mit übertragen wird. Grundsätzlich muß man zwischen zwei Schutzarten unterscheiden.

Da gibt es einerseits den sogenannten Hardware-Schutz. Das ist beispielsweise ein elektronisches Bauteil, das in den User-Port oder auch den Joystickport gesteckt wird. In diesem "elektronischen Schlüssel" befindet sich dann ein integrierter Schaltkreis, der die in dem Programm zu verarbeitenden Daten in einer ganz bestimmten Form umsetzt oder logisch verschiebt. Da das Innenleben eines solchen "Dongls" nicht erkennbar ist - er ist mit einer Kunststoffmasse vergossen - bedarf es schon eines fundierten Wissens auf dem Gebiet der Elektronik und manch guter Meßgeräte, um der Art und Weise, wie die Daten verschlüsselt werden, auf die Schliche zu kommen. Für einen Privatmann ist diese Methode jedoch gänzlich ungeeignet, da er nicht über die nötigen Geräte zur Herstellung solcher Hardware-Sicherungen verfügt. Aber trösten Sie sich, denn abgesehen von den hohen Herstellungskosten hat auch ein Hardwareschutz den Nachteil, daß er früher oder später überlistet wird.

Wir wollen uns auf den folgenden Seiten mit der Software-Lösung dieses Problems beschäftigen. Es bleibt hier sicherlich noch anzumerken, daß die folgenden Zeilen bestenfalls eine

Einführung in diese Thematik darstellen. Wenn Sie sich für dieses Gebiet interessieren, so sei an dieser Stelle auf das Buch "64 TIPS & TRICKS, Band 2" verwiesen.

Unser Kopierschutz besteht aus zwei Teilen. Der erste Teil ist die Vorbereitung, der zweite stellt dann, in modular aufgebauter Form, den eigentlichen Schutz dar. Dieses Modul läßt sich problemlos in neue, aber auch in schon geschriebene Programme implementieren. Ein sehr großer Vorteil unserer Lösung ist dadurch gegeben, daß es derzeit nur zwei Kopierprogramme auf dem einschlägigen Markt gibt, die in der Lage sind, unsere Hürde zu überwinden. Damit sind Sie zumindest gegen manchen "Gelegenheitstäter" einigermaßen abgesichert.

Bei allen Versuchen mit der nachfolgenden Lösung sollten Sie sich aber unbedingt an zwei Richtlinien halten:

Bevor Sie die Funktion der beiden Programme austesten, sollten Sie diese unbedingt vorher abspeichern, da die Gefahr relativ groß ist, daß etwas schief läuft. Das hat zur Folge, daß alle Erfassungsarbeit umsonst war. Verwenden Sie bei allen Tests und eventuellem späteren Praxiseinsatz grundsätzlich nur leere Disketten, da das Vorbereitungsprogramm einen Diskettenblock verändert, ohne darauf Rücksicht zu nehmen, ob dieser vielleicht schon von normalen Daten belegt ist.

Hier nun die Vorbereitungsroutine:

```
10 REM **********************************
20 REM SCHEINBLOCK VORBEREITEN
30 REM **********************************
40 PRINT CHR$(147)
50 PRINT"   WOLLEN SIE EINEN BESTIMMTEN BLOCK"
60 PRINT"   VORBEREITEN (J/N)"
70 GET A$ : IF A$<>"J" AND A$<>"N" THEN GOTO 70
80 IF A$ = "N" THEN TT = 18 : SS = 2 : GOTO 200
90 PRINT
100 INPUT"   TRACK    ";TT
```

```
110 IF TT>35 THEN PRINT"EXISTIERT NICHT"; CHR$(145);
    CHR$(145) : GOTO 100
120 PRINT"                              "
130 SX = 16
140 IF TT   18 THEN SX = 20 *****
150 IF TT   25 THEN SX = 18 *****
160 IF TT   31 THEN SX = 17 *****
170 INPUT"   SEKTOR   ";SS
180 IF SS>SX THEN PRINT"EXISTIERT NICHT"; CHR$(145);
    CHR$(145) : GOTO 170
190 PRINT"                              "
200 OPEN 1,8,15 : OPEN 2,8,2," "
210 PRINT 1,"B-P:"2;0
220 INPUT"   CODEWORT ";C$
230 IF LEN(C$)   15 THEN PRINT"ZU LANG";
CHR$(145); CHR$(145) : GOTO 220
240 PRINT 2,C$
250 PRINT 1,"U2:"2;0;TT;SS
260 CLOSE2
270 PRINT 1,"B-F:"0;TT;SS
280 CLOSE1
290 PRINT"   FERTIG !" : END
```

Was wird durch diese kleine Programm erreicht?

Sie können sich entscheiden, ob Sie einen von Ihnen bestimmten Block auf der Diskette vorbereiten wollen oder nicht. Wenn Sie sich für "N" entscheiden, so notieren Sie sich bitte die Werte: Spur 18, Sektor 2. Wenn Sie einen anderen Block vorbereiten, so müssen Sie sich die von Ihnen vergebenen Werte merken. Das Programm prüft automatisch, ob Ihre Eingaben überhaupt möglich sind und weist Sie dann auf eventuelle Fehler hin. Das geschieht alles in den Zeilen 10 bis 190. In Zeile 200 wird dann der Kommandokanal und ein Datenkanal zur Floppy eröffnet. Anschließend werden Sie aufgefordert, ein "Codewort" einzugeben. Auch dieses müssen Sie sich unbedingt merken. In der hier vorgestellten Programmversion darf das Codewort nicht länger als 15 Zeichen sein.

Das Codewort wird dann in den Datenbuffer 2 geschrieben, der seinerseits danach als kompletter Block an die vorgegebene Stelle auf der Diskette geschrieben wird. Abschließend wird der eben aufgebrachte Block in der BAM der Diskette als "frei" gekennzeichnet. Das ist einer der Gründe, warum diese Schutzfunktion für viele Kopierprogramme unüberwindbar ist.

Einen Haken hat die Sache allerdings. Wenn Sie die vom Programm vorgegebene Adresse verwenden, so laufen Sie Gefahr, diese wieder zu löschen, wenn Sie mehr als 24 Programme auf Ihrer vorbereiteten Diskette abspeichern. Haben Sie selbst die Adresse bestimmt, dann läßt es sich nicht vorhersagen, ob und wann Ihr Block wieder zu einem normalen Datenblock wird.

Kommen wir jetzt zum zweiten Teil unseres Sicherungspakets, der sogenannten Blockabfrage.

```
10 REM **********************************
20 REM SCHEINBLOCK ABFRAGEN
30 REM **********************************
40 PRINT"DAS IST EINE DEMONSTRATION"
50 PRINT"FUER DIESEN KOPIERSCHUTZ"
60 INPUT"CODEWORT ";C$
70 TT = 18 : SS = 2
80 GOSUB 60000
90 END
60000 OPEN1,8,15 : OPEN2,8,2," "
60010 PRINT 1,"U1:"2;0;TT;SS
60020 INPUT 2,A$
60030 CLOSE1 : CLOSE2
60040 IF A$ = C$ THEN RETURN
60050 SYS 64738
```

Dieses kleine Demonstrationsprogramm muß auf der vorher vorbereiteten Diskette abgespeichert werden, da es sonst nicht funktionstüchtig ist. Die eigentliche Sicherungsfunktion finden Sie in den Unterprogrammzeilen 60000 bis 60050. Somit stehen

Ihnen alle BASIC-Zeilen unter 60000 für Ihr eigenes Programm zur Verfügung. Zu beachten ist dabei nur, daß Sie Ihr Codewort in der Variablen C$ ablegen, bevor Sie das Unterprogramm anspringen. Ebenso sollten Sie bei der Kontrolle, ob das richtige Codewort auf der Diskette ist (in Zeile 60040), dort den direkten Code eintragen.

```
60040 IF A$ = "Ein Codewort" THEN RETURN
```

Ferner gilt es zu beachten, daß die Adresse auf der Diskette entweder in den Variablen TT (für Spur) und SS (für Sektor) abgelegt ist oder daß auch sie direkt in dem Block-Lese-Befehl der Zeile 60010 eingetragen wird.

```
60010 PRINT 1,"U1:"2;0;18;2
```

In der Zeile 60050 des Unterprogramms können Sie dann wahlfrei entscheiden, was geschehen soll, wenn bei der Kontrolle keine Übereinstimmung erzielt wurde. In unserem Fall wird ein Reset des Rechners ausgelöst. Das heißt, der Rechner wird wieder in seinen Einschaltzustand versetzt. Das Programm im Rechner ist dann selbstverständlich verloren. Sie sehen, daß sich Ihnen mit dieser Methode ungeahnte Möglichkeiten zur Sicherung Ihres geistigen Eigentums bieten.

Selbstverständlich ist es auch denkbar, daß Sie mehrere Blöcke auf Ihrer Diskette in der beschriebenen Art und Weise mit einem Schlüssel vorbereiten.

Um zu sehen, daß unser Schutz auch wirklich funktioniert, sollten Sie einmal hergehen und das von Ihnen vergebene Codewort falsch eintragen.

An dieser Stelle muß aber noch darauf hingewiesen werden, daß es relativ einfach ist, die ganze Schlüsselabfrage aus dem Pro-

gramm zu entfernen. Das läßt sich vermeiden, indem Sie Ihr Programm compilieren oder einen guten LIST-Schutz für ein BASIC-Programm verwenden.

5.2 Hardcopy auf Epson Druckern

Nachdem wir nun die hochauflösende Grafik kennengelert haben, und auch schon eine Hardcopy-Routine für den Seikosha GP-100VC, möchte ich Ihnen nun noch extra eine Hardcopyroutine für alle Epson und kompatiblen Druckern vorstellen. Diese Hardcopyroutine druckt eine Hardcopy, die im Bereich von 8192 bis 16383 liegt ($2000 - $4000). Wie Sie sehen werden, sofern Sie über einen Epson-Drucker verfügen, lassen sich mit dieser Hardcopyroutine viel bessere Hardcopies erstellen, als man Sie je auf einem Seikosha-Drucker erstellen könnten. Doch nun zunächst das Assemblerprogramm:

```
120:  c000                .opt p1,oo
130:  c000                *=  49152
140:                ;
150:  c000 48             pha
160:  c001 a9 00          lda  #<8192
160:  c003 a0 20          ldy  #>8192
170:  c005 85 fa          sta  $fa
170:  c007 84 fb          sty  $fb     ; basis adresse setzen
180:                ;
190:  c009 68             pla
190:  c00a 48             pha
200:  c00b a0 01          ldy  #1      ; sekundaeradresse
210:  c00d 20 ba ff       jsr  $ffba   ; setpara
220:  c010 a9 00          lda  #0
230:  c012 20 bd ff       jsr  $ffbd   ; setnam
240:  c015 20 c0 ff       jsr  $ffc0   ; open
250:  c018 68             pla
260:  c019 48             pha
270:  c01a aa             tax
280:  c01b 20 c9 ff       jsr  $ffc9   ; chkout
290:                ;
```

```
300:   c01e a9 1b          lda  #27     ; escape "1" senden,
310:   c020 20 d2 ff       jsr  $ffd2   ; dadurch zeilenabstand
320:   c023 a9 31          lda  #"1"    ; setzen
330:   c025 20 d2 ff       jsr  $ffd2
340:                ;
350:   c028 20 45 c0       jsr  hardcopy  ; hardcopy drucken
360:                ;
370:   c02b a9 1b          lda  #27     ; escape "2" senden,
380:   c02d 20 d2 ff       jsr  $ffd2   ; dadurch zeilenabstand
390:   c030 a9 32          lda  #"2"    ; normalisieren
400:   c032 20 d2 ff       jsr  $ffd2
410:   c035 20 cc ff       jsr  $ffcc   ; clrch
420:   c038 68             pla
430:   c039 4c c3 ff       jmp  $ffc3   ; close
440:                ;
450:   c03c 00       bytes  .byt %00000000
460:   c03d 00              .byt %00000000
470:   c03e 00              .byt %00000000
480:   c03f 00              .byt %00000000
490:   c040 00              .byt %00000000
500:   c041 00              .byt %00000000
510:   c042 00              .byt %00000000
520:   c043 00              .byt %00000000
530:                ;
540:   c044 00       spalte .byt 0       ; spaltenzaehler
550:                ;
560:   c045 a2 18    hardcopy ldx #24     ; zeilenzaehler setzen
570:   c047 a9 0d    lab1     lda #13
570:   c049 20 d2 ff          jsr  $ffd2   ; zeilenvorschub
580:   c04c a0 04             ldy  #4
590:   c04e b9 75 c0 loop     lda  drcktab,y  ; steuerzeichen
600:   c051 20 d2 ff          jsr  $ffd2   ; ausgeben
610:   c054 88                dey
610:   c055 10 f7             bpl  loop
620:                ;
630:   c057 a9 27             lda  #39     ; spaltenzaehler
640:   c059 8d 44 c0          sta  spalte  ; setzen
650:   c05c a0 00    lab2     ldy  #0      ; y-register loeschen
660:   c05e 20 7a c0          jsr  sendacht  ; 8ter block ausgeben
670:   c061 a5 fa             lda  $fa     ; $fa/$fb = $fa/$fb + 8
```

```
680:   c063 18                clc
690:   c064 69 08             adc  #8
700:   c066 90 02             bcc  ok1
710:   c068 e6 fb             inc  $fb
720:   c06a 85 fa     ok1     sta  $fa
730:   c06c ce 44 c0          dec  spalte  ; spalte=spalte-1
740:   c06f 10 eb             bpl  lab2
750:   c071 ca                dex
760:   c072 10 d3             bpl  lab1
770:   c074 60                rts
780:                  ;
790:   c075 02 80 4c drcktab   .byt 2,128,76,27,9
800:                  ;
810:                  ;*** drehen und setzen eines 8ter blocks ***
820:                  ;
830:   c07a 8a        sendacht txa
830:   c07b 48                 pha
830:   c07c 98                 tya
830:   c07d 48                 pha
840:   c07e a9 80              lda  #$80
850:   c080 8d b7 c0          sta  maske
860:   c083 a2 07             ldx  #7
870:   c085 a0 00     hrdloop1 ldy  #0
880:   c087 b1 fa     hrdloop2 lda  ($fa),y
890:   c089 2d b7 c0          and  maske
900:   c08c 38                sec
910:   c08d d0 01             bne  ok2
920:   c08f 18                clc
930:   c090 3e 3c c0 ok2      rol  bytes,x
940:   c093 c8                iny
950:   c094 c0 08             cpy  #8
960:   c096 d0 ef             bne  hrdloop2
970:   c098 4e b7 c0          lsr  maske
980:   c09b ca                dex
980:   c09c 10 e7             bpl  hrdloop1
990:                  ;
1000:  c09e a2 07             ldx  #7      ; bytes ausgeben
1010:  c0a0 bd 3c c0 outlab   lda  bytes,x
1020:  c0a3 20 d2 ff          jsr  $ffd2
1020:  c0a6 20 d2 ff          jsr  $ffd2
```

```
1020:  c0a9 a9 00            lda  #0
1020:  c0ab 9d 3c c0         sta  bytes,x
1030:  c0ae ca               dex
1040:  c0af 10 ef            bpl  outlab
1050:  c0b1 68               pla
1050:  c0b2 a8               tay
1050:  c0b3 68               pla
1050:  c0b4 aa               tax
1050:  c0b5 60               rts
1060:                ;
1070:  c0b6 00       temp    .byt 0
1080:  c0b7 00       maske   .byt 0
```

Als nächstes der BASIC-Lader für die, diè keinen Assembler oder Monitor zur Verfügung haben.

```
100 rem ********************************
110 rem ***                            ***
120 rem ***    hardcopy epson          ***
130 rem ***   ----------------         ***
140 rem ***                            ***
150 rem ********************************
160 :
170 for x=49152 to 49335
180 :read a:poke x,a
190 :p=p+a
200 next x
210 :
220 if p<>22834 then print "fehler in datas !!!":stop
230 :
240 end
250 :
50000 data  72,169,  0,160, 32,133,250
50010 data 132,251,104, 72,160,  1, 32
50020 data 186,255,169,  0, 32,189,255
50030 data  32,192,255,104, 72,170, 32
50040 data 201,255,169, 27, 32,210,255
```

```
50050 data 169, 49, 32,210,255, 32, 69
50060 data 192,169, 27, 32,210,255,169
50070 data  50, 32,210,255, 32,204,255
50080 data 104, 76,195,255,  0,  0,  0
50090 data   0,  0,  0,  0,  0,  0,162
50100 data  24,169, 13, 32,210,255,160
50110 data   4,185,117,192, 32,210,255
50120 data 136, 16,247,169, 39,141, 68
50130 data 192,160,  0, 32,122,192,165
50140 data 250, 24,105,  8,144,  2,230
50150 data 251,133,250,206, 68,192, 16
50160 data 235,202, 16,211, 96,  2,128
50170 data  76, 27,  9,138, 72,152, 72
50180 data 169,128,141,183,192,162,  7
50190 data 160,  0,177,250, 45,183,192
50200 data  56,208,  1, 24, 62, 60,192
50210 data 200,192,  8,208,239, 78,183
50220 data 192,202, 16,231,162,  7,189
50230 data  60,192, 32,210,255, 32,210
50240 data 255,169,  0,157, 60,192,202
50250 data  16,239,104,168,104,170, 96
50260 data   0,  0
```

Nun zur genauen Anwendung. Bevor Sie eine Hardcopy erstellen, müssen Sie zunächst die logische Filenummer und die Geräteadresse übermitteln. Dies geschieht mit den Befehlen:

```
poke780,logfil
poke781,gerätadr
```

Anschließend brauchen Sie nur noch zu warten, bis die Routine Ihre Hardcopy fertiggestellt hat.

5.3 Schreibschutzdetektor

Es ist oft vorteilhaft, feststellen zu können, ob eine im Disk-Drive verwendete Diskette mit einem mechanischen Schreibschutz versehen ist.

Die umseitig folgende "Detector"-Routine gibt Ihnen die Möglichkeit, dies direkt über eine Software-Lösung festzustellen.

```
10 PRINT "(CLR)DISK-SCHREIBSCHUTZ DETECTOR"
20 OPEN 15,8,15
30 : PRINT#15, "M-R";CHR$(0);CHR$(28)
40 :    GET D$
50 :    D=ASC(D$+CHR$(0)) AND 16
55 :
60 :    M$="KEIN SCHREIBSCHUTZ!"
70 :    IF D=0 THEN M$="SCHREIBSCHUTZ VORHANDEN!"
80 : PRINT M$
90 CLOSE 15
100 END
```

Wenn Sie dieses Programm ausführen, wird ermittelt, ob die aktuelle Disk mit einem Schreibschutz-Klebestreifen versehen ist oder nicht. Sie haben so beispielsweise die Möglichkeit, innerhalb eines Programms festzustellen, ob Datenfiles geSAVEd werden können, wenn also kein Schreibschutz vorhanden ist.

5.4 Disk-Versiegler

Wo wir uns gerade mit dem Schreibschutz befaßt haben, möchte ich Ihnen nun eine weitere effektive Routine präsentieren, anhand derer man einen Schreibschutz per Software realisieren kann.

Das lästige und doch etwas umständliche Aufkleben und später wieder Entfernen der Schreibschutz-Klebestreifen können Sie nun ein für allemal vergesssen.

Geben Sie zunächst wieder einmal die entsprechende Schreibschutz-Routine ein:

```
10 PRINT "(CLR)DISK-VERSIEGELN
20 X$=CHR$(1)
30 Z$=" 2 0 18 0"
35 :
40 OPEN 1,8,15,"I"
50 :   PRINT#1,"M-W";X$;X$;X$;CHR$(65)
60 :   PRINT#1,"U1";Z$
70 :   PRINT#1,"B-P 2 2"
80 :      OPEN 2,8,2,"#"
90 :        PRINT#2,"(F6-Taste)"
100 :     CLOSE 2
110 :  PRINT#1,"U2";Z$
120 :  PRINT#1,"I"
130 CLOSE 1
```

Mit der Ausführung des obigen Programms wird die sich in dem Disk-Slot befindende Diskette mit einem Software-Schreibschutz versehen.

Sie ist damit versiegelt und nur noch lesefähig. Programme und Datenfiles können nun nicht mehr auf die Disk geschrieben werden oder versehentlich gelöscht werden.

Lediglich eine komplette Disketten-Formatierung, die aber auch alle gespeicherten Daten löscht, macht die Disk wieder schreibfähig.

5.5 Disk-Entsiegler

Wie der Titel bereits andeutet, gibt es neben der eben beschriebenen, doch recht "brutalen" Methode, eine versiegelte Diskette über die Formatierung wieder schreibfähig zu machen,

noch ein eleganteres und sinnvolleres Verfahren, den Schreibschutz wieder zu entfernen.

Die Entsiegler-Routine ist ähnlich wie das Schreibschutz-Programm konzipiert:

```
10 PRINT "(CLR)DISK-ENTSIEGELN
20 X$=CHR$(1)
30 Z$=" 2 0 18 0"
35 :
40 OPEN 1,8,15,"I"
50 :   PRINT#1,"M-W";X$;X$;X$;CHR$(65)
60 :   PRINT#1,"U1";Z$
70 :   PRINT#1,"B-P 2 2"
80 :      OPEN 2,8,2,"#"
90 :        PRINT#2,"A"
100 :     CLOSE 2
110 :  PRINT#1,"U2";Z$
120 :  PRINT#1,"I"
130 CLOSE 1
```

Nachdem Sie diese Routine mit RUN gestartet haben, wird der durch die VERSIEGLER-Routine generierte Schreibschutz wieder gelöscht.

Die Disk ist damit wieder entsiegelt und schreibfähig. Alle gespeicherten Daten und Files bleiben bei diesem Vorgang selbstverständlich erhalten.

Die beiden Routinen sind erheblich komfortabler zum Erzeugen eines Schreibschutzes als das herkömmliche Verfahren mittels der Schreibschutz-Klebestreifen.

5.6 Programme als USR-/Seq-Files

Wenn Sie ein Programm wie gewöhnlich mit

```
SAVE "PROGRAMM",8
```

speichern, so erhalten Sie in der Directory den Eintrag:

```
"PROGRAMM" PRG
```

Sie können dasselbe Programm aber auch als sequentielles File SAVEn. Geben Sie dazu ein:

```
SAVE "PROGRAMM,S",8
```

Der Directory-Eintrag lautet dann wie folgt:

```
"PROGRAMM" SEQ
```

Ein derart gespeichertes Programm wird später nicht mehr geladen, es sei denn, Sie geben den Befehl

```
LOAD "PROGRAMM,S",8
```

ein.

Dadurch erhalten Sie ein hohes Maß an Sicherheit, daß derart geSAVEte Programme nicht von Unbefugten geladen werden. Unter einem SEQ-FILE vermutet niemand ein herkömmliches BASIC- oder MC-Programm.

Auf ähnliche Weise wie SEQ-Files können Sie Programme auch als USR in der Directory ablegen. Dies geschieht über

```
SAVE "PROGRAMM,U",8
```

Wieder geladen werden kann ein derartiges Programm mittels

```
LOAD "PROGRAMM,U",8
```

Dies können Sie dafür einsetzen, um einen Hinweis zu geben, daß ein Programm beispielsweise nur per

```
LOAD "XXXX",8,1
```

geladen werden muß.

Viele Computer reservieren USR-Files ausschließlich für MC-Files. Dies können Sie nun auch in der beschriebenen Weise auf den C-64 und auch dem C-128 realisieren. Neben dem Aspekt des Programmschutzes erhält man so zudem eine größere Übersicht aller auf einer Diskette vorhandenen Programme und deren Inhalt.

5.7 Floppy-Reset

Es ist in einigen Situationen recht hilfreich, über eine Softwarelösung ein Reset des Disk-Drives zu erzeugen.

Dies geschieht, indem man den Befehl "UJ" in den Command-Kanal sendet. Führen Sie dazu den folgenden Einzeiler aus:

```
10 OPEN 15,8,15: PRINT#15,"UJ": CLOSE 15
```

Dieser wird fast immer problemlos einen Reset generieren. Sollte er jedoch keinen Effekt zeigen, so reagiert die Floppy nicht mehr auf Befehle.

In diesem Fall hilft nur noch ein Aus- und wieder Einschalten des Drives, nachdem man die Diskette kurzfristig vorher entnommen hat.

5.8 Close all

Immer wieder liest man, der Befehl

```
SYS 65511
```

schließe alle Files. Das ist jedoch ein (weit verbreiteter) Irrtum. Er bewirkt lediglich, daß der Computer die geöffneten Files vergißt. Er gibt jedoch kein Signal aus, daß die Files geschlossen werden sollen.

Dieser feine Unterschied kann katastrophale Folgen nach sich ziehen.

Um alle oder aber Files aus einem bestimmten Bereich ordnungsgemäß und korrekt zu schließen, gibt es jedoch eine einfache Methode.

Statt wie bisher die Befehlskette

```
CLOSE 1
CLOSE 2
CLOSE 3
CLOSE 4
...
CLOSE 15
```

einzugeben, genügt eine einzige kurze Routine, die genau dasselbe bewirkt:

```
10 FOR I=1 TO PEEK(152):CLOSE PEEK(601):NEXT I
```

Dadurch werden alle geöffneten Files geschlossen.

Dabei beinhaltet die Adresse 152 die Anzahl der momentan offenen Files. Übrigens entspricht der Befehl SYS 65511 genau der Anweisung POKE 152,0, d.h. er bewirkt lediglich das Vergessen der offenen Files.

Unter der Adresse 601 ist die Filenummer gespeichert.

```
LOAD "*"
```

Dieser Befehl dient bekanntermaßen dazu, das erste Programm auf der Diskette zu laden. In der Praxis zeigt sich jedoch, daß meistens nicht das erste Programm, sondern oft das Programm, welches als letztes angesprochen wurde, geladen wird.

In einigen Fällen ist die Auswahl des zu ladenden Programms auch rein zufällig.

Um diesen Fehler auszuschalten müssen Sie statt des herkömmlichen Befehls

```
LOAD "*"
```

einfach die Anweisung

```
LOAD ":*"
```

eingeben.

Der Doppelpunkt bewirkt dabei, daß auch wirklich das erste Programm auf einer Diskette geladen wird. Einfach, aber es wirkt!

5.9 Komma-Files

Bei Experimenten mit Disk-Files können einfache Programmfehler bereits zu merkwürdigen Effekten führen.

So kann beispielsweise ein File namens

```
","
```

in der Directory auftauchen. Dieses wiedersetzt sich allen Versuchen, es zu löschen oder umzubenennen.

Ist in der Directory sonst kein anderes File mit nur einem Zeichen als Titel vorhanden, so kann er mit dem Filenamen "?" anstelle von "," gescratched oder umbenannt werden.

Beinhaltet die Directory jedoch bereits ein weiteren File mit einem nur ein Zeichen langen Titel, so müssen sie dieses zunächst mit einem längeren Filetitel versehen.

5.10 Scratch-File-Recover

Wenn Sie ein File mit der Anweisung

```
OPEN 1,8,15,"S:PROGRAMM"
CLOSE 1
```

irrtümlich aus dem Directory-Eintrag gelöscht haben, so können Sie das Programm dennoch vollständig retten.

Geben Sie dazu den Befehl

```
LOAD "**",8
```

bzw.

```
LOAD "*A",8
```

wobei A den Anfangsbuchstaben des gelöschten Programms darstellt, ein.

Das geSCRATCHte Programm wird dann vollständig neu in den Speicher geladen. Voraussetzung ist jedoch, daß nach dem SCRATCH kein RESET der FLOPPY erfolgte.

5.11 Directory-Kill

Viele kopiergeschützte Programme besitzen keine bzw. eine leere Directory. Es wird also kein Programm oder File-Eintrag angezeigt.

Mit der hier vorgestellten Routine haben Sie nun die Möglichkeit, auch Ihre eigenen Files vor den Augen Unbefugter zu verbergen.

Führen Sie dazu die folgende Routine aus:

```
10 OPEN 1,8,3,"#"
20 PRINT#1,CHR$(20);CHR$(20);CHR$(20);
30 PRINT#1,CHR$(0) ;CHR$(0) ;CHR$(0)
40 OPEN 2,8,15,"B-P3,144"
50 PRINT#2,"U2:3,0,18"
60 PRINT#2,"I"
70 CLOSE 1
80 CLOSE 2
```

Diese schreibt drei Nullen an den Anfang der Directory, welche das Programmende anzeigen. Dadurch wird der LIST-Vorgang frühzeitig abgebrochen und beendet.

Die Auswirkungen der Routine sind dadurch jedoch noch weitreichender. Abgesehen von dem gelöschten Directory ist die Diskette damit automatisch schreibgeschützt. Eine Änderung dieses Zustandes ist dann nur noch über eine komplette Neu-Formatierung möglich.

5.12 Selektiver Directory-Schutz

Die obrige Routine zum Löschen bzw. Verbergen der kompletten Directory ist äußerst effektiv.

Sie können aber auf eine noch einfachere Weise Teile von Directory-Einträgen gezielt unsichtbar machen. Obwohl derartige Titel nicht mehr im Inhaltsverzeichnis auftauchen, können sie immer noch geladen werden.

Angenommen Sie haben die folgende Directory vorliegen:

```
0   "MANIPULIERTE DIR" TEIL1

10  "AUTO-PROGR. A"     PRG
7   "AUTO-PROGR. B"     PRG
4   "FILES"             SEQ
2   "DIRECTORY-KILL"    PRG
3   "FLOPPY VERSIEGLER" PRG
3   "ENTSIEGLER"        PRG
50  "INTIPRO"           PRG
72  "EXPERTENSYSTEM"    PRG
91  "ELIZA"             USR

440 BLOCKS FREE.
```

Wollen Sie nun beispielsweise verhindern, daß Unbefugte die letzten drei File-Einträge sehen können, so muß lediglich der vorherige Eintrag,

```
3   "ENTSIEGLER"        PRG
```

etwas geändert werden.

Geben Sie dazu die folgenden Befehle ein:

```
OPEN 1,8,15,"R:ENTSIEGLER"+CHR$(0)+CHR$(0)+CHR$(0)+
"=ENTSIEGLER"
CLOSE 1
```

Hier wird der Filename "ENTSIEGLER" umbenannt. An das Ende des Eintrags werden drei CHR$(0)-Zeichen gehängt. Diese sind wie bereits vorher erwähnt, für den Computer ein Signal dafür, daß die Directory beendet ist.

Daher wird die Directory nach dem so präperiertem File nicht mehr gelistet. Die modifizierte Directory sieht in unserem Beispiel folgendermaßen aus:

```
0   "MANIPULIERTE DIR" TEIL 1

10  "AUTO-PROGR. A"      PRG
7   "AUTO-PROGR. B"      PRG
4   "FILES"              SEQ
2   "DIRECTORY-KILL"     PRG
3   "FLOPPY VERSIEGLER"  PRG
3   "ENTSIEGLER"         PRG

440 BLOCKS FREE
```

Man kann diesen Vorgang auch wahlweise rückgängig machen. Dies geschieht in unserem konkreten Beispiel über diese Anweisungen:

```
OPEN 1,8,15,"R:ENTSIEGLER=ENTSIEGLER"+CHR$(0)+
CHR$(0)+CHR$(0)
CLOSE 1
```

5.13 Directory-Steuercodes

Sinnvoll verwendbare Steuercodes in einer Directory sind beispielsweise

```
FARBEN:       CHR$(5)  ..... WEIß
              CHR$(28) ..... ROT
              CHR$(30) ..... GRÜN
              CHR$(31) ..... BLAU
              CHR$(144)..... SCHWARZ
              CHR$(156)..... PURPUR
              CHR$(158)..... GELB
              CHR$(159)..... CYAN


SONDERCODES: CHR$(8)  ..... C= TASTE BLOCKIERT
              CHR$(9)  ..... C= TASTE ENTRIEGELT
              CHR$(14) ..... UMSCHALTUNG KLEINSCHRIFT
              CHR$(18) ..... REVERSE AN
              CHR$(19) ..... HOME
              CHR$(20) ..... DELETE
              CHR$(142)..... UMSCHALTUNG GROßSCHRIFT
              CHR$(147)..... CLEAR
```

Diese Steuercodes lassen sich am sinnvollsten in die Directory-Einträge integrieren.

Sie können beispielsweise das Programm "ROM-QUICKCOPY" statt einfach mit

```
SAVE "QUICKCOPY",8
```

auch über die folgende Anweisung SAVEn:

```
SAVE CHR$(147)+CHR$(5)+CHR$(8)+CHR$(14)+"QUICKCOPY",8
```

Alle Steuercodes werden beim nächsten Laden über den Directory-Eintrag ausgeführt.

Das bedeutet für unser Beispiel, daß schon beim Ladevorgang ein CLEAR über CHR$(147) ausgeführt wird. Die CHR$(5) setzt die Vordergrundfarbe auf weiß, CHR$(8) blockiert die C=Taste und mittels CHR(14) wird die Kleinschrift eingeschaltet.

5.14 Sondereinträge

Sie können den einzelnen Directory-Eintrag noch um eine beliebige Zusatzinformation erweitern. Mit der hier beschriebenen Methode sind beispielsweise die folgenden Titel möglich:

```
62  "EXPERT" (C)OH'85   PRG
```

Hier wurde eine Mitteilung (C) in den Titel eingearbeitet. Das besondere hierbei ist jedoch, daß das Programm wie üblich mittels

```
LOAD "EXPERT",8
```

geladen werden kann.

Ein weiteres Beispiel sind eingearbeitete Hinweise. Der Titel

```
45  "ARCADE" ,8,1  PRG
```

zeigt sofort an, daß das Programm mit LOAD "ARCADE",8,1 geladen werden muß.

Bei

```
112  "GAME" JOY 1   PRG
```

wissen Sie sofort durch den Eintrag JOY 1, daß das Spiel mit dem Titel "GAME" den Joystickport Nummer 1 verwendet.

Die Anwendungsmöglichkeiten sind vielfältig.

Befassen wir uns nun mit der Realisierung. Um das Programm "AUTO-PRG", das sich bereits auf DISKETTE befinden muß, mit einer (C)-Mitteilung zu versehen, müssen Sie folgendes eingeben:

```
OPEN 1,8,15,"R:AUTO-PRG"+CHR$(160)+" (C)OH'85"+"=AUTO-PRG"
CLOSE 1
```

Der Disketten-Eintrag lautet dann so:

```
34  "AUTO-PRG" (C)OH'85   PRG
```

Geladen wird dieses Programm wie üblich mit dem Titel "AUTO-PRG",8.

Der neue Eintrag wird folgendermaßen generiert: Die Rename-Routine funktioniert normalerweise nach dem bekannten Prinzip:

```
"R:NEU=ALT"
```

In unserem Fall haben wir den neuen Namen zusammengesetzt und ein CHR$(160) Zeichen eingefügt. Dieses Zeichen, das dem SHIFT SPACE entspricht, zeigt dem Computer an, daß der Filename zu Ende ist. Dadurch wird das zweite Anführungszeichen ausgegeben.

Die dann folgenden Kommentare werden zwar gespeichert, aber beim Programmtitel nicht berücksichtigt, das Programm kann also wie üblich geladen werden.

Die Anwendungen dieses Tricks sind vielfältig. Neben den hier gezeigten Beispielen von Copyright-Kommentaren, Joystickport-Anzeigen oder Ladeinformationen können beispielsweise auch laufende Programmversionsnummern, Up-date Datumsanzeigen etc. in der beschriebenen Weise indirekt in die Titel eingefügt werden.

5.15 Tape Speed-up

An dieser Stelle sollen noch ganz kurz einige TAPE-TIPS vorgestellt werden:

1) Wie Sie sicher wissen, werden Cassetten-Programme doppelt gespeichert. Dieses Verfahren wird gewählt, um so eventuelle Ladefehler abzufangen.

Dies können Sie ausnutzen, um die Ladezeit nahezu zu halbieren. Durch die Anweisungen

```
POKE 45,PEEK(831)
POKE 46,PEEK(832)
```

und

```
CLR
```

die Sie in das zu ladende Programm am Anfang einfügen müssen - haben Sie die Möglichkeit, nachdem die erste Programm-Kopie geladen ist, die STOP-Taste zu betätigen.

Die obrigen drei Anweisungen bewirken, daß dieser Einbruch in den Ladevorgang ohne Folgen bleibt und keine Errors im Programm generiert werden.

2) Wenn der Computer nach der FOUND-Meldung für einige Zeit stoppt, so können Sie diese Pause ganz einfach durch einen Druck der CTRL-, der SPACE- oder der Commodore-Taste beenden. Dasselbe bewirkt ein Druck der Feuer-Taste eines an Port 1 angeschlossenen Joysticks.

3) Sie können auch MC-Routinen im Cassetten- Buffer ablegen und anschließend Programme über die Datasette laden. Geben Sie dazu den

```
POKE 179,4
```

vor dem Zugriff auf das Tape ein.

Der Computer benutzt dann den Screen-Memory-Bereich anstelle des Cassetten-Buffers. Alle MC-Routinen im Bereich von 828 bis 1019 bleiben so vollständig erhalten.

Kapitel 6

Befehlserweiterungen

6.1 Der erweiterte SYS-Befehl

Sollten Sie selbst BASIC und Assembler programmieren, so werden Sie bestimmt selbst wissen, wie umständlich es ist, Parameter von BASIC aus an das Maschinenprogramm zu übergeben. Meistens sieht es so aus, daß die jeweiligen 8-Bit Werte in irgenwelche Speicherregister vor dem Aufruf des BASIC-Programm gePOKEd werden müssen. Schlimmer wird es dann bei 16-Bit-Werten, bei denen zuerst High- und LOW-Byte errechnet werden müssen. Ganz aus ist es dann, wenn Strings an das untergeordnete Assemblerprogramm übergeben werden müssen.

Doch bietet das Betriebssystem uns hier auch noch andere Möglichkeiten an, die wir im Folgenden nutzen werden. Es besteht nämlich die Möglichkeit durch den SYS-Befehl alle nur denkbaren Parameter an ein Assemblerprogramm zu übergeben. Dieses geschieht durch die verschiedensten Betriebssystem-Unterroutinen, die nun im Folgenden anhand von zahlreichen Beispielen erklärt werden sollen:

Zunächst einmal die wichtigste Routine. Dies ist die Routine, die ein Komma aus dem BASIC-Text holt. Dadurch wird es uns möglich, weitere Parameter, die natürlich durch ein Komma getrennt werden müssen, aus dem BASIC-Text zu holen. Ein Komma muß aus folgendem Grund zur Trennung der Parameter stehen, der Aufruf

```
SYS 49152 20
```

würde vom BASIC-Interpreter fälschlicher weise als ein

```
SYS 4915220
```

interpretiert werden, was natürlich zu einem "ILLEGAL QUANTITY ERROR" führen würde. Einen Aufruf wie zum Beispiel

```
SYS 49152,20
```

kann der BASIC-Interpreter jedoch problemlos verarbeiten, sofern eine weitere Voraussetzung erfüllt ist. Nachdem wir nämlich ein Komma geholt haben, müssen wir auch noch den entsprechenden Wert aus dem BASIC-Text lesen. Dieser Wert, der ja zunächst in ASCII-Form vorliegt, wird in das sogenannte "Fließkommaformat" umgewandelt. Dies ist das Format, in dem alle Zahlen im Commodore 64 vorliegen. Anschließend wird dieses Fließkommaformat, das in dem sogenannten "Fließkommaakku" (FAC) vorliegt, in ein 8-Bit-Intergerformat umgewandelt (0-255), der wiederum nun mit den ganz normalen Prozessorregistern bearbeitet werden kann. Zum Glück bietet der BASIC-Interpreter schon zahlreiche Routinen, mit denen wir die verschiedensten Umwandlungen vornehmen können. Doch nun ein Beispiel. Wir wollen einen Funktionsaufruf entwickeln, mit dem wir Hintergrund- und Rahmenfarbe setzen können. Ein derartiger Aufruf sollte etwa wie folgt aussehen:

```
SYS ad,hf,rf
```

Wobei "ad" die Startadresse unseres neuen Befehls ist und "hf" und "rf" die Werte für die entsprechenden Farben sind.

Hier nun das entsprechende Assemblerprogramm:

```
*=49152         ; Startadresse
;
jsr $aefd       ; Prüft auf Kommata
jsr $b79e       ; Holt ein Byte ins X-Register
stx 53281       ; setzt die Rahmenfarbe
jsr $aefd       ; Prüft auf weiteres Komma
jsr $b79e       ; Holt Byte-Wert ins X-Register
stx 53280       ; setzt die Hintergrundfarbe
rts             ; Rücksprung ins BASIC-Programm
```

Die beiden wichtigsten Routinen sind also

```
getbyt      - $b79e
chkcom      - $aefd
```

Mit diesen Routinen läßt sich schon eine ganze Menge anfangen. Aber es gibt noch weitere Routinen, die Ihnen das Programmieren von Funktions- und Befehlserweiterungen erheblich erleichtern. Grundvoraussetzung dafür allerdings ist, daß Sie sich etwas mit der Assemblerprogrammierung auskennen. Fangen wir also an mit der Auflistung (aller) nötigen Routinen zur Befehlserweiterungsprogrammierung. Dabei beschränken diese Funktionen sich oft nicht nur auf den Aufruf eines einzigen Unterprogramms:

getbyt

Anwendung: JSR $B79E

Funktion: Holt einen Ein-Byte-Wert aus dem BASIC-Text und übergibt diesen dem X-Register. Sollte der Wert die Größe einer 8-Bit-Zahl überschreiten, so wird automatisch die Fehlermeldung "ILLEGAL QUANTITY ERROR" ausgeben.

Ausgabeparam.: X-Register

getadr (I)

Anwendung:
```
JSR $AD8A ; holt numerischen Wert
JMP $B7F7 ; FAC nach Adressformat
```

Funktion: Holt einen 16-Bit-Wert aus dem BASIC-Text und übergibt diesen den Speicherstellen $14/$15 im Format LOW/HIGH. Sollte der Wert die Größe einer 16-Bit-Zahl überschreiten, so wird automatisch die Fehlermeldung "ILLEGAL QUANTITY ERROR" ausgegeben.

Ausgabeparam.: $14 (LOW-Byte)
$15 (HIGH-Byte)

getadr (II)

Anwendung: JSR $B7EB

Funktion: Holt einen 16-Bit-Wert aus dem BASIC-Text und übergibt diesen den Speicherstellen $14/$15 im Format HIGH/LOW. Anschließend prüft die Routine noch auf Komma und holt schließlich einen 8-Bit-Wert in das X-Register. Sollte der 1. Wert die Größe einer 16-Bit-Zahl oder der 2. Wert die Größe einer 8-Bit-Zahl übersteigen, so wird automatisch die Fehlermeldung "ILLEGAL QUANTITY ERROR" ausgegeben.

Ausgabeparam.: $14 (LOW-Byte Wert 1)
$15 (HIGH-Byte Wert 1)
X-Register (Wert 2)

chkslsr

Anwendung: JSR $AEF7

Funktion: Prüft auf ")" (Klammer zu) im aktuellen BASIC-Text. Sollte ein anderes oder gar kein Zeichen folgen, so wird automatisch die Fehlermeldung "SYNTAX ERROR" ausgegeben.

Ausgabeparam.: keine

chkslsl

Anwendung: JSR $AEFA

Funktion: Prüft auf "(" (Klammer auf) im aktuellen BASIC-Text. Sollte ein anderes oder gar kein Zeichen folgen, so wird automatisch die Fehlermeldung "SYNTAX ERROR" ausgegeben.

Ausgabeparam.: keine

chkcom

Anwendung: JSR $AEFD

Funktion: Prüft auf "," (Komma) im BASIC-Text. Sollte ein anderes oder gar kein Zeichen folgen, so wird automatisch die Fehlermeldung "SYNTAX ERROR" ausgegeben.

Ausgabeparam.: keine

getchar

Anwendung:
```
LDA #"zeichen"
jsr $AEFF
```

Funktion: Prüft auf das Zeichen, das sich im Akku des Prozessors befindet. Sollte ein anderes oder gar kein Zeichen folgen, so wird automatisch die Fehlermeldung "SYNTAX ERROR" ausgegeben.

chkcom (II)

Anwendung:

```
        jsr $0079       ; laufendes Zeichen holen
        cmp #","        ; Vergleich mit Komma
        bne fer         ; wenn nicht, dann fertig
        JSR $0073       ; nächsten Zeichen holen
        lda #0          ; Zeroflag setzen
fer     rts             ; Routine beenden
```

Funktion: Prüft auf ein Komma im laufenden BASIC-Text. Bei Übereinstimmung wird das Zeroflag gesetzt, ansonsten bleibt es gelöscht. Dadurch haben Sie die Möglichkeit, Parameter auch nur bei Bedarf anzugeben oder wegzulassen.

Ausgabeparam: Zeroflag. Bei gesetzem Zeroflag existiert ein Komma und wird auch aus dem laufenden BASIC-Text gelesen.

getstr

Anwendung:

```
clc             ; Übertrag löschen
jsr $AD9E       ; beliebigen Ausdruck holen
jsr $A6A3       ; auf String prüfen
```

Funktion: Prüft auf eine Zeichenkette, die durch Anführungszeichen (") eingeklammert ist, im laufenden BASIC-Text. Die Stringlänge wird im Akku übergeben. Zeiger auf den String bilden die Speicherstellen $22/$23 im Format LOW/HIGH. Beim Holen des Strings

wird auch auf freien Speicherplatz geachtet. Dabei wird, wenn nicht genügend Speicher vorhanden ist, die Fehlermeldung "OUT OF MEMORY ERROR" ausgegeben. Sollte der String die Länge von 255 Zeichen überschreiten, so wird die Fehlermeldung "STRING TOO LONG ERROR" ausgegeben.

Ausgabeparam.: Akku - Länge des Strings
$22/$23 - Zeiger auf String

combyt

Anwendung:

```
        jsr $aefd      ; holt Komma
        jsr $0079      ; holt laufendes Zeichen
        cmp.#","       ; vergleicht mit ","
        beq end        ; folgt nicht, dann zu "end"
        jsr $b79e      ; holt Byte ins X-Register
        rts            ; Unterroutine zu Ende
end     ldx #0         ; X-Register mit 0 laden
        rts            ; Unterroutine zu Ende
```

Funktion: Prüft auf Komma und holt Ein-Byte-Wert. Folgt dem Komma direkt ein weiteres Komma, so wird das X-Register mit 0 geladen und die Routine beendet. Dadurch werden Funktionsaufrufe wie

SYS 49152,,,4,20

anstelle des längeren

SYS 49152,0,0,0,4,20

möglich

Ausgabeparam.: X-Register

gnumvr

Anwendung:

```
jsr $B08B      ; holt Variable
ldy wert       ; Y-Register mit 8-Bit-Wert
                 laden
jsr $B3A2      ; nach FAC-Format wandeln
ldx $47        ; Zeiger kopieren
stx $49
ldx $48
stx $4a
jsr $BBD0      ; FAC nach Variable
                 übertragen
```

Funktion: Holt eine Variable aus dem laufenden BASIC-Text, und schreibt in diese den Wert, der sich im Y-Register (s.o.) befindet. Ein anschließender Ausdruck der Variable ergibt dann den Wert, der sich im Y-Register befindet.

Ausgabeparam.: keine

gstrvr

Anwendung:

```
jsr $b08b      ; holt Variable
lda $47        ; Zeiger kopieren
sta $49
lda $48
sta $4a
lda #(stpr     ; Lage der Stringpara-
ldy #)stpr     ; meter übertragen
sta $64
sty $65
jsr $AA52      ; String in Variable kopieren
               ;
stpr .byt laenge,(lage,)lage
```

Funktion: Holt eine Stringvariable aus dem BASIC-Text und ordnet ihr den String mit der Länge "laenge" an der Lage "lage" zu. Auf diese Art können von Assembler aus Strings an BASIC übergeben werden.

Ausgabeprarm.: keine

6.2 Keine negativen Zahlen bei der FRE-Funktion

Ist es Ihnen auch schon aufgefallen, daß sich Ihr Commodore 64 beim Einschalten zwar mit "38911 Bytes" free meldet, auf den Befehl

```
PRINT FRE(0)
```

aber mit -26627 antwortet?

Hat man eine negative Zahl erhalten, muß man erst 2 hoch 16 gleich 65536 addieren, um den richtigen Wert zu erhalten. Dies ist zwar nicht tragisch, aber lästig. Woran liegt dies nun?

Dazu müssen wir uns im ROM-Listing die entsprechende Stelle ansehen. (Adresse $B37D: 64 intern). Dort wird, nachdem die

nicht mehr gebrauchten Strings entfernt und deren Speicherplatz wieder freigegeben wird (Garbage Collection), der freie Speicherplatz berechnet: Beginn der Strings ($33/$34) minus Ende der Arrays ($31/$32). Dieser 16-Bit-Integerwert wird nun ins Fließkommaformat gewandelt und zurückgegeben. Hier liegt nun der Fehler. Der Integerwert wird als Zahl mit Vorzeichen behandelt analog den Integervariablen (%), die nur Werte von -32768 bis 32767 haben können. Werden diese Zahlen jedoch als positive Werte betrachtet, können sie im Bereich von 0 bis 65535 liegen. Bei den frühen Commodore-Rechnern war der freie Speicherplatz nie größer als 32767 Bytes, so daß dieser Fehler nie auftrat. Wir müssen jetzt die FRE-Routine so ändern, daß bei der Umwandlung in Fließkommazahlen von einem positiven Integerwert ausgegangen wird, wie dies zum Beispiel auch bei der Ausgabe von Zeilennummern der Fall ist, die ja bekanntlich größer als 32767 sein können.

Dies sind die Änderungen, die erforderlich sind. Hier haben wir den zusätzlichen Code in einen unbenutzten Bereich des BASIC-Interpreters gelegt.

```
B38D   4C 55 BF   JMP $BF55
B390   EA         NOP

BF55   A5 34      LDA $34
BF57   E5 32      SBC $32
BF59   A2 00      LDX #$00
BF5B   86 0D      STX $0D
BF5D   85 62      STA $62
BF5F   84 63      STY $63
BF61   A2 90      LDX #$90
BF63   4C 49 BC   JMP $BC49
```

Die Werte lassen sich mit einer kleinen POKE-Schleife setzen.

```
100 FOR I=40960 TO 49151
110 POKE I, PEEK(I) : NEXT
120 A=11*4096+3*256+8*16+13
130 FOR I=A TO A+3
140 READ X : POKE I,X :NEXT
150 A=11*4096+15*256+5*16+5
160 FOR I=A TO A+16
170 READ X : POKE I,X : NEXT
180 POKE 1,54
200 DATA 76,85,191,234
210 DATA 165,52,229,50,162,0,134,13,133,98,132,99
220 DATA 162,144,76,73,188
```

6.3 Die Erweiterung des MID$-Befehls

Sie kennen sicher alle die MID$-Funktion, mit der ein Teil einer Zeichenkette isoliert werden kann. Das folgende Programmbeispiel

```
100 A$ = "TESTSTRING"
110 B$ = MID$(A$,5,3)
120 PRINT B$
```

führt zum Ausdruck

```
STR
```

Nachfolgend finden Sie nun eine Erweiterung des MID$-Befehls, die nicht nur das AUSLESEN von Teilen eines Strings ermöglicht, sondern auch das ERSETZEN von Teilen eines Strings. Mit dem neuen Befehl ist z.B. folgende Programmierung möglich:

```
100 A$ = "TESTSTRING"
110 MID$ (A$,5,3) = "123"
120 PRINT A$
```

Hier werden 3 Zeichen ab Position 5 durch die Zeichenfolge "123" ersetzt; als Ergebnis entsteht

```
TEST123ING
```

Ohne MID$-Befehl müßten Sie den String in einzelne Teile zerlegen und neu zusammen setzen:

```
100 A$ = "TESTSTRING"
110 A$ = LEFT$(A$,4) + "123" + MID$(A$,7,3)
120 PRINT A$
```

Dieser Befehl ist z.B. dann sehr nützlich, wenn Sie in einem Datensatz einzelne Teile (Datenfelder) ersetzen wollen. Dazu definieren Sie einen String in der Länge des Datensatzes und ersetzen mit dem neuen MID$-Befehl die Werte der einzelnen Datenfelder.

Der Befehl wird wieder mit einem kleinen Maschinenprogramm realisiert.

```
90:    033c                .opt p1
; mid$ als pseudovariable
;
; mid$(stringvariable,position,laenge) = stringausdruck
; mid$(stringvariable,position) = stringausdruck
;
200:   033c        midcode  =   $ca
210:   033c        execut   =   $308    ; vector für statement
                                          ausführen
```

```
220:  033c             chrget   =   $73
230:  033c             chrgot   =   chrget + 6
240:  033c             execold  =   $a7e7
250:  033c             varnam   =   $45
255:  033c             varadr   =   $49
260:  033c             descrpt  =   $64
270:  033c             teststr  =   $ad8f
280:  033c             getvar   =   $b08b
290:  033c             setstr   =   $aa52
300:  033c             chkauf   =   $aefa   ; klammer auf
310:  033c             chkzu    =   $aef7   ; klammer zu
320:  033c             chkcom   =   $aefd   ; komma
325:  033c             test     =   $aeff
330:  033c             getbyt   =   $b79e
340:  033c             frmevl   =   $ad9e
350:  033c             illquan  =   $b248
352:  033c             frestr   =   $b6a3
355:  0003                      *=   3
360:  0004             laenge   *=   *+1
370:  0005             position *=   *+1
372:  0007             varstr   *=   *+2
375:  0007             gleich   =   $b2
378:  0007             zeig2    =   $50
390:  033c                      *=   828
400:  033c a9 47       init     lda  #<midtest
410:  033e a0 03                ldy  #>midtest
420:  0340 8d 08 03             sta  execut
430:  0343 8c 09 03             sty  execut+1
440:  0346 60                   rts
450:  0347 20 73 00 midtest     jsr  chrget
460:  034a c9 ca                cmp  #midcode  ; code für mid$ ?
470:  034c f0 06                beq  midstr  ; ja
480:  034e 20 79 00             jsr  chrgot
490:  0351 4c e7 a7             jmp  execold  ; normales statement
                                             ausführen
500:  0354 20 73 00 midstr      jsr  chrget  ; nächstes zeichen
505:  0357 20 fa ae             jsr  chkauf  ; klammer auf
510:  035a 20 8b b0             jsr  getvar  ; variable holen
520:  035d 85 64                sta  descrpt
530:  035f 84 65                sty  descrpt+1
```

```
535:  0361 85 49          sta  varadr
535:  0363 84 4a          sty  varadr+1
540:  0365 20 a3 b6       jsr  frestr
545:  0368 a0 00          ldy  #0
545:  036a b1 64          lda  (descrpt),y
545:  036c 48             pha          ; länge
545:  036d f0 2e          beq  ill
550:  036f 20 52 aa       jsr  setstr  ; string in ram übertragen
560:  0372 a0 01          ldy  #1
560:  0374 b1 49          lda  (varadr),y
560:  0376 85 05          sta  varstr  ; variablenadresse merken
570:  0378 c8             iny
570:  0379 b1 49          lda  (varadr),y
570:  037b 85 06          sta  varstr+1
600:  037d 20 fd ae       jsr  chkcom
610:  0380 20 9e b7       jsr  getbyt  ; position holen
620:  0383 8a             txa
630:  0384 f0 17          beq  ill
650:  0386 ca             dex
650:  0387 86 04          stx  position
660:  0389 20 79 00       jsr  chrgot
660:  038c c9 29          cmp  #")"    ; ausdruck zu ende ?
665:  038e d0 04          bne  next
665:  0390 a9 ff          lda  #$ff    ; max. länge
665:  0392 d0 0c          bne  store
670:  0394 20 fd ae next  jsr  chkcom
670:  0397 20 9e b7       jsr  getbyt  ; länge holen
680:  039a 8a             txa
690:  039b d0 03          bne  *+5
700:  039d 4c 48 b2 ill   jmp  illquan
710:  03a0 85 03    store sta  laenge
715:  03a2 68             pla
715:  03a3 38             sec
715:  03a4 e5 04          sbc  position
717:  03a6 c5 03          cmp  laenge
717:  03a8 b0 02          bcs  ok
717:  03aa 85 03          sta  laenge
720:  03ac 20 f7 ae ok    jsr  chkzu   ; klammer zu
730:  03af a9 b2          lda  #gleich
770:  03b1 20 ff ae       jsr  test
```

```
780:   03b4 20 9e ad          jsr  frmevl  ; ausdruck holen
790:   03b7 20 a3 b6          jsr  frestr
800:   03ba a0 02             ldy  #2
800:   03bc b1 64             lda  (descrpt),y
800:   03be 85 51             sta  zeig2+1
800:   03c0 88                dey
800:   03c1 b1 64             lda  (descrpt),y
800:   03c3 85 50             sta  zeig2
810:   03c5 88                dey
810:   03c6 b1 64             lda  (descrpt),y
820:   03c8 f0 d3             beq  ill     ; null, dann fehler
840:   03ca c5 03             cmp  laenge
850:   03cc b0 02             bcs  ok1
860:   03ce 85 03             sta  laenge
870:   03d0 a5 05    ok1      lda  varstr
880:   03d2 18                clc
880:   03d3 65 04             adc  position
910:   03d5 85 05             sta  varstr
910:   03d7 90 02             bcc  *+4
920:   03d9 e6 06             inc  varstr+1
940:   03db a4 03             ldy  laenge
950:   03dd 88       loop     dey
950:   03de b1 50             lda  (zeig2),y  ; zeichen aus
                                                stringausdruck
960:   03e0 91 05             sta  (varstr),y ; in stringvariable
                                                übertragen
970:   03e2 c0 00             cpy  #0
970:   03e4 d0 f7             bne  loop
980:   03e6 4c ae a7          jmp  $a7ae   ; zur interpreterschleife
```

Nach dem Eingeben des Programms können Sie mit dem Befehl

```
SYS 828
```

Ihre Befehlserweiterung initialisieren. Das nachfolgend wiedergegebene BASIC-Ladeprogramm macht diese Initialisierung automatisch.

```
100 for i = 828 to 1000
110 read x : poke i,x : s=s+x : next
120 data 169, 71,160,  3,141,  8,  3,140,  9,  3, 96, 32
130 data 115,  0,201,202,240,  6, 32,121,  0, 76,231,167
140 data  32,115,  0, 32,250,174, 32,139,176,133,100,132
150 data 101,133, 73,132, 74, 32,163,182,160,  0,177,100
160 data  72,240, 46, 32, 82,170,160,  1,177, 73,133,  5
170 data 200,177, 73,133,  6, 32,253,174, 32,158,183,138
180 data 240, 23,202,134,  4, 32,121,  0,201, 41,208,  4
190 data 169,255,208, 12, 32,253,174, 32,158,183,138,208
200 data   3, 76, 72,178,133,  3,104, 56,229,  4,197,  3
210 data 176,  2,133,  3, 32,247,174,169,178, 32,255,174
220 data  32,158,173, 32,163,182,160,  2,177,100,133, 81
230 data 136,177,100,133, 80,136,177,100,240,211,197,  3
240 data 176,  2,133,  3,165,  5, 24,101,  4,133,  5,144
250 data   2,230,  6,164,  3,136,177, 80,145,  5,192,  0
260 data 208,247, 76,174,167
270 if s <> 19273 then print "fehler in datas !!" : end
280 sys 828 : print "ok !"
```

Als Beispiel und Test für die neue Funktion kann folgendes Programm dienen:

```
100 DIM A$(10)
110 FOR I = 1 TO 10
120 A$(I) = "TESTSTRING"
130 NEXT
140 FOR I = 1 TO 10
150 MID$ (A$(I),I,1) = MID$("1234567890",I,1)
160 NEXT
170 FOR I = 1 TO 10
180 PRINT A$(I)
190 NEXT
```

Als Ergebnis erhalten Sie 10 Strings, wobei im ersten String das erste Zeichen durch eine "1" ersetzt wurde, im zweiten String steht als 2. Zeichen einen "2" usw.:

```
1ESTSTRING
T2STSTRING
TE3TSTRING
TES4STRING
TEST5TRING
TESTS6RING
TESTST7ING
TESTSTR8NG
TESTSTRI9G
TESTSTRINO
```

6.4 INSTR- und STRING$-Funktion

Viele andere BASIC-Rechner haben zwei sehr nützliche Stringfunktionen, die dem 64er fehlen. Die eine Funktion, meist STRING$ genannt, erzeugt mit einem Befehl einen String beliebiger Länge mit einem vorgegebenen ASCII-Code, die andere Funktion, oft INSTR genannt, prüft ob ein String in einem anderen enthalten ist.

Mit der Kenntnis des BASIC-Interpreters und der Stringverwaltung des Commodore 64 ist es nun möglich, auf dem 64er auch diese Funktionen zu implementieren. Als Bezeichnung benutzen wir die schon vorhandenen Befehlsworte 'POS' und 'STR$' des 64ers, die wir zur Unterscheidung durch ein vorangestelltes "!" kennzeichnen.

Die INSTR-Funktion hat dann folgende Syntax:

```
I = !POS( A$,B$,P)
```

Dabei ist A$ der String, den Sie durchsuchen wollen, B$ ist der String, dessen Vorkommen in A$ Sie überprüfen wollen und P ist die Position, ab der im String A$ gesucht wird. Das Ergebnis wird der Variablen I zugewiesen und ist gleich Null, wenn der gesuchte String nicht gefunden wurde. Kommt der zweite String im ersten vor, so enthält I die Position, an der er gefunden wurde. Die Angabe der Position P ist wahlfrei; wird sie nicht angegeben, so wird von Anfang an gesucht. An Stelle von Variablen können Sie überall auch Ausdrücke verwenden oder die Funktion selbst in Ausdrücken, z.B. einer Print-Liste benutzen.

Hier einige Beispiel zur Anwendung:

```
PRINT !POS("ABCDEFGHIJK","D")
 4

IF !POS(A$,"J") THEN PRINT "GEFUNDEN"
A$ = "TESTSTRING"
PRINT !POS(A$,"T")
 1

X = !POS(A$,"T",5) : PRINT X
 6
```

Die STRING$-Funktion wird folgendermaßen benutzt:

```
A$ = !STR$ (L,B)
```

oder

```
A$ = !STR$ (L,B$)
```

Hierbei ist A$ der String, den Sie erzeugen wollen. L ist die Länge, die der erzeugte String erhalten soll und B ist der ASCII-Code des Strings. Wird anstelle von B ein String verwendet, wird der ASCII-Code des ersten Zeichens dieses String benutzt. Folgende Beispiele demonstrieren die Anwendung

```
PRINT !STR$(20,65)
AAAAAAAAAAAAAAAAAAAA

A$ = !STR$(10,"*") : PRINT A$
**********
```

Das Maschinenprogramm ist im freien RAM-Bereich abgelegt und beginnt bei Adresse 51200.

```
100:   c800                    .opt p1
; instring-funktion
;
135:   c800                    *=   $c800
140:   c800          chkauf    =    $aefa
150:   c800          chkzu     =    $aef7
160:   c800          chkcom    =    $aefd
170:   c800          frmevl    =    $ad9e
180:   c800          chkstr    =    $ad8f
190:   c800          frestr    =    $b6a3
200:   c800          yfac      =    $b3a2
205:   c800          chrget    =    $73
210:   c800          chrgot    =    chrget+6
220:   c800          getbyt    =    $b79b
226:   c800          integer   =    $b1aa
```

```
229:  c800             descrpt  =   $64
230:  c800             stradr   =   $62
231:  c800             adr2     =   $fb
232:  c800             adr1     =   $fb+2
233:  c800             len1     =   3
234:  c800             len2     =   4
235:  c800             anzahl   =   5
236:  c800             start    =   6
237:  c800             typflag  =   13
238:  c800             strcode  =   $c4
240:  c800             illquan  =   $b248
241:  c800             syntax   =   $af08
242:  c800             poscode  =   $b9
243:  c800             vector   =   $30a
245:  c800             temp     =   len1
248:  c800 a9 0b               lda  #<testin
248:  c802 a0 c8               ldy  #>testin
248:  c804 8d 0a 03            sta  vector
248:  c807 8c 0b 03            sty  vector+1
248:  c80a 60                  rts
250:  c80b a9 00       testin  lda  #0
250:  c80d 85 0d               sta  typflag
250:  c80f 20 73 00            jsr  chrget
251:  c812 c9 21               cmp  #"!"
251:  c814 f0 06               beq  test2
251:  c816 20 79 00            jsr  chrgot
251:  c819 4c 8d ae            jmp  $ae8d
252:  c81c 20 73 00 test2      jsr  chrget
252:  c81f c9 b9               cmp  #poscode
252:  c821 f0 0a               beq  instr
252:  c823 c9 c4               cmp  #strcode
252:  c825 d0 03               bne  *+ 5
253:  c827 4c b1 c8            jmp  string
253:  c82a 4c 08 af            jmp  syntax
300:  c82d 20 73 00 instr      jsr  chrget
300:  c830 20 fa ae            jsr  chkauf  ; klammer auf
310:  c833 20 9e ad            jsr  frmevl  ; ausdruck holen
320:  c836 20 8f ad            jsr  chkstr  ; string testen
330:  c839 a5 64               lda  descrpt
330:  c83b 48                  pha          ; stringadresse auf stack
```

```
340:  c83c a5 65          lda  descrpt+1
340:  c83e 48             pha
350:  c83f 20 fd ae       jsr  chkcom  ; komma
360:  c842 20 9e ad       jsr  frmevl  ; zweiter string
380:  c845 20 a3 b6       jsr  frestr
390:  c848 f0 64          beq  ill     ; länge = 0
400:  c84a 85 04          sta  len2
410:  c84c 86 fb          stx  adr2
420:  c84e 84 fc          sty  adr2+1
430:  c850 68             pla
430:  c851 a8             tay
440:  c852 68             pla          ; erste stringadresse
450:  c853 20 aa b6       jsr  frestr+7
460:  c856 f0 56          beq  ill
470:  c858 85 03          sta  len1
480:  c85a 86 fd          stx  adr1
490:  c85c 84 fe          sty  adr1+1
500:  c85e a2 00          ldx  #0      ; default für 3. parameter
510:  c860 20 79 00       jsr  chrgot
520:  c863 c9 2c          cmp  #","
530:  c865 d0 07          bne  l1
540:  c867 20 9b b7       jsr  getbyt
550:  c86a 8a             txa          ; startposition
560:  c86b f0 41          beq  ill
570:  c86d ca             dex
580:  c86e 86 06    l1    stx  start   ; startposition im string
590:  c870 20 f7 ae       jsr  chkzu
600:  c873 a5 03          lda  len1
610:  c875 38             sec          ; len2 > len1
620:  c876 e5 04          sbc  len2
630:  c878 90 28          bcc  end     ; ergebnis 0 ?
640:  c87a 69 00          adc  #0
650:  c87c 85 05          sta  anzahl  ; der verschiebungen
660:  c87e a5 06          lda  start
660:  c880 18             clc          ; adresse 1 plus
                                         startposition
670:  c881 65 fd          adc  adr1
670:  c883 85 fd          sta  adr1
680:  c885 90 02          bcc  *+4
690:  c887 e6 fe          inc  adr1+1
```

```
700:   c889 a0 00    l2       ldy  #0
710:   c88b b1 fb    l3       lda  (adr2),y
720:   c88d d1 fd             cmp  (adr1),y  ; zeichen vergleichen
730:   c88f d0 0b             bne  l5        ; ab nächster position suchen
740:   c891 c8                iny
740:   c892 c4 04             cpy  len2      ; schon alle zeichen string2
                                               getestet ?
750:   c894 90 f5             bcc  l3
760:   c896 a4 06             ldy  start
770:   c898 c8                iny
780:   c899 4c a2 b3 l4       jmp  yfac      ; ergebnis
790:   c89c e6 06    l5       inc  start
800:   c89e c6 05             dec  anzahl
810:   c8a0 d0 04             bne  l6        ; noch nicht fertig ?
820:   c8a2 a0 00    end      ldy  #0        ; nicht gefunden, null
830:   c8a4 f0 f3             beq  l4
840:   c8a6 e6 fd    l6       inc  adr1
850:   c8a8 d0 df             bne  l2        ; string2 adresse erhöhen
860:   c8aa e6 fe             inc  adr1+1
870:   c8ac d0 db             bne  l2
880:   c8ae 4c 48 b2 ill      jmp  illquan
            ;
            ; string$-funktion
            ;
900:   c8b1 20 73 00 string   jsr  chrget
900:   c8b4 20 fa ae          jsr  chkauf    ; klammer auf
910:   c8b7 20 9e b7          jsr  getbyt+3
920:   c8ba 8a                txa
920:   c8bb 48                pha            ; länge merken
930:   c8bc 20 fd ae          jsr  chkcom
940:   c8bf 20 9e ad          jsr  frmevl
950:   c8c2 24 0d             bit  typflag
960:   c8c4 30 0c             bmi  str       ; string
970:   c8c6 20 aa b1          jsr  integer
980:   c8c9 a5 64             lda  descrpt   ; highbyte
990:   c8cb d0 e1             bne  ill       ; > 255
1000:  c8cd a5 65             lda  descrpt+1  ; low-byte, länge
1010:  c8cf 4c db c8          jmp  str2
1020:  c8d2 20 82 b7 str      jsr  $b782     ; setstr, typflag auf
                                               numerisch
```

```
1030:  c8d5 f0 d7            beq  ill     ; länge null
1040:  c8d7 a0 00            ldy  #0
1050:  c8d9 b1 22            lda  ($22),y ; erstes zeichen
1060:  c8db 85 03    str2    sta  temp
1070:  c8dd 68               pla          ; länge
1080:  c8de 20 7d b4         jsr  $b47d   ; frestr
1090:  c8e1 a8               tay
1100:  c8e2 f0 07            beq  str3
1110:  c8e4 a5 03            lda  temp
1120:  c8e6 88       loop    dey
1120:  c8e7 91 62            sta  (stradr),y  ; string erzeugen
1130:  c8e9 d0 fb            bne  loop
1140:  c8eb 20 ca b4 str3     jsr  $b4ca   ; string in descriptorstack
                                           bringen
1150:  c8ee 4c f7 ae          jmp  chkzu
```

Hier wieder für 'Nicht-Maschinen-Programmierer' ein Ladeprogramm in BASIC, das auch bereits die Initialisierung übernimmt.

```
100 for i = 51200 to 51440
110 read x : poke i,x : s=s+x : next
120 data 169, 11,160,200,141, 10,  3,140, 11,  3, 96,169
130 data   0,133, 13, 32,115,  0,201, 33,240,  6, 32,121
140 data   0, 76,141,174, 32,115,  0,201,185,240, 10,201
150 data 196,208,  3, 76,177,200, 76,  8,175, 32,115,  0
160 data  32,250,174, 32,158,173, 32,143,173,165,100, 72
170 data 165,101, 72, 32,253,174, 32,158,173, 32,163,182
180 data 240,100,133,  4,134,251,132,252,104,168,104, 32
190 data 170,182,240, 86,133,  3,134,253,132,254,162,  0
200 data  32,121,  0,201, 44,208,  7, 32,155,183,138,240
210 data  65,202,134,  6, 32,247,174,165,  3, 56,229,  4
220 data 144, 40,105,  0,133,  5,165,  6, 24,101,253,133
230 data 253,144,  2,230,254,160,  0,177,251,209,253,208
240 data  11,200,196,  4,144,245,164,  6,200, 76,162,179
250 data 230,  6,198,  5,208,  4,160,  0,240,243,230,253
260 data 208,223,230,254,208,219, 76, 72,178, 32,115,  0
270 data  32,250,174, 32,158,183,138, 72, 32,253,174, 32
280 data 158,173, 36, 13, 48, 12, 32,170,177,165,100,208
```

4.) Seite laden

Durch diese Funktion laden Sie eine abgespeicherte Grafik wieder in den Rechnerspeicher. Dort kann diese dann weiterverarbeitet werden.

5.) Programm löschen

Nachdem Sie diesen Punkt angewählt haben, wird das Programm gelöscht. Die bis dahin gespeicherten PRINT-Zeilen bleiben erhalten und können so in ein fertiges Programm übernommen werden.

1.24 Rückkehr ins BASIC-Programm nach einem LIST-Befehl

Wenn man in ein BASIC-Programm einen LIST-Befehl einbaut, so springt das Programm nach der Ausführung des LIST-Befehls immer in den Direkt-Modus zurück. Dies ist dann störend, wenn Sie z.B. Zeilen ausgeben wollen, die etwa eine Funktionsdefinition mit DEF FN enthalten um die jeweilige Funktion auf dem Bildschirm anzuzeigen. Ebenso könnten Sie kein Programmlisting von einem Programm aus mehrfach ausgeben, ebenso nicht im Direkt-Modus in einer Schleife, z.B.

```
FOR I=1 TO 2 : LIST : NEXT
```

Hier läßt sich folgendermaßen Abhilfe schaffen: Sie ersetzen am Ende der LIST-Funktion den Sprung zum BASIC-Warmstart durch einen RETURN-Befehl. Zusätzlich müssen Sie vor dem Aufruf der LIST-Funktion den Zeiger auf den Programmtext retten, da er während der LIST-Funktion verändert wird.

Dazu benötigen wir eine kleine Routine, die diese Aufgaben durchführt und wieder zum BASIC-Interpreter zurückspringt. Da dies sowieso ein kleines Maschinenprogramm erfordert,

Um den AUTO-Modus zu beenden, drücken Sie einfach nur RETURN ohne eine weitere Eingabe. Wollen Sie später weitere Programmzeilen eingeben, so genügt die Eingabe von

```
SYS 828
```

ohne weitere Angaben. Es wird dann automatisch bei der nächsten folgenden Zeilenummer weitergemacht, bei der vorher abgebrochen wurde. Natürlich können Sie jederzeit Startwert und Inkrement ändern, indem Sie diese Werte beim SYS-Befehl mit angeben.

Es folgt nun das Listing des Maschinenprogramms, das im Cassettenpuffer abgelegt ist sowie anschließend wieder ein Ladeprogramm in BASIC.

```
150:  033C                  *=  828     ; Kassettenpuffer
152:  033C          LO      =   $14
154:  033C          HI      =   LO+1
156:  033C          FAC     =   $62     ; Fließkommaakku
158:  033C          CR      =   13      ; carriage return
160:  033C          LINE    =   251     ; Zeile
162:  033C          INCR    =   LINE +2 ; Inkrement
164:  033C          INTFLOAT =  $BC49   ; Integer nach Fließkomma
166:  033C          FLOATASCI=  $BDDD   ; Fließkomma nach ASCII
170:  033C          VECTOR  =   $302    ; ZEILENEINGABE
172:  033C          INPUT   =   $FFCF
174:  033C          PRINT   =   $FFD2
176:  033C          BUFFER1 =   $101
178:  033C          BUFFER2 =   $200
180:  033C          MAINLOOP =  $A486
182:  033C          GOON    =   $A569
183:  033C          WEITER  =   $A576
184:  033C          CHRGOT  =   $79
186:  033C          OLDVEC  =   $A483
187:  033C          GETPAR  =   $B7EB
189:  033C          CHKCOM  =   $AEFD
190:  033C 20 79 00         JSR CHRGOT  ; weitere Zeichen ?
190:  033F F0 10            BEQ LO      ; nein
```

```
200:   0341 20 FD AE          JSR  CHKCOM  ; Komma ?
200:   0344 20 EB B7          JSR  GETPAR  ; Parameter holen
210:   0347 86 FD             STX  INCR
220:   0349 A5 14             LDA  LO      ; und abspeichern
220:   034B 85 FB             STA  LINE
230:   034D A5 15             LDA  HI
230:   034F 85 FC             STA  LINE+1
240:   0351 A9 5C    LO       LDA  #< AUTO
240:   0353 8D 02 03          STA  VECTOR  ; Eingabevektor setzen
250:   0356 A9 03             LDA  #> AUTO
250:   0358 8D 03 03          STA  VECTOR+1
260:   035B 60                RTS
;
300:   035C 20 62 03 AUTO     JSR  AUTONUM
310:   035F 4C 86 A4          JMP  MAINLOOP
;
330:   0362 A5 FB    AUTONUM  LDA  LINE
330:   0364 A6 FC             LDX  LINE+1
340:   0366 85 63             STA  FAC+1   ; Zeilenummer
340:   0368 86 62             STX  FAC
340:   036A A2 90             LDX  #$90
340:   036C 38                SEC
350:   036D 20 49 BC          JSR  INTFLOAT; nach Fließkomma
360:   0370 20 DD BD          JSR  FLOATASCI  ; nach ASCII
370:   0373 A2 00             LDX  #0
380:   0375 BD 01 01 L1       LDA  BUFFER1,X ; Ziffern holen
390:   0378 F0 09             BEQ  L2
400:   037A 9D 00 02          STA  BUFFER2,X ; in BASIC-Puffer
400:   037D 20 D2 FF          JSR  PRINT   ; und ausgeben
410:   0380 E8                INX
410:   0381 D0 F2             BNE  L1
420:   0383 A5 FB    L2       LDA  LINE    ; Zeilenummer
420:   0385 18                CLC          ; erhöhen
420:   0386 65 FD             ADC  INCR
420:   0388 85 FB             STA  LINE
420:   038A 90 02             BCC  L3
420:   038C E6 FC             INC  LINE+1
425:   038E A9 20    L3       LDA  #" "    ; Leerzeichen ausgeben
425:   0390 20 D2 FF          JSR  PRINT
425:   0393 20 CF FF          JSR  INPUT
```

```
430:   0396 C9 0D          CMP  #CR     ; leere Eingabe
440:   0398 F0 03          BEQ  L4      ; ja
450:   039A 4C 69 A5       JMP  GOON    ; weitermachen
460:   039D A5 FB    L4    LDA  LINE
460:   039F E5 FD          SBC  INCR    ; Zeilenummer für nächsten
460:   03A1 85 FB          STA  LINE    ; Zugriff erniedrigen
470:   03A3 B0 02          BCS  L5
470:   03A5 C6 FC          DEC  LINE +1
480:   03A7 A9 83    L5    LDA  #< OLDVEC
480:   03A9 A0 A4          LDY  #> OLDVEC
490:   03AB 8D 02 03       STA  VECTOR  ; alten Vektor
490:   03AE 8C 03 03       STY  VECTOR+1 ;wieder herstellen
500:   03B1 4C 76 A5       JMP  WEITER
```

```
100 FOR I = 828 TO 947
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA  32,121,  0,240, 16, 32,253,174, 32,235,183,134
130 DATA 253,165, 20,133,251,165, 21,133,252,169, 92,141
140 DATA   2,  3,169,  3,141,  3,  3, 96, 32, 98,  3, 76
150 DATA 134,164,165,251,166,252,133, 99,134, 98,162,144
160 DATA  56, 32, 73,188, 32,221,189,162,  0,189,  1,  1
170 DATA 240,  9,157,  0,  2, 32,210,255,232,208,242,165
180 DATA 251, 24,101,253,133,251,144,  2,230,252,169, 32
190 DATA  32,210,255, 32,207,255,201, 13,240,  3, 76,105
200 DATA 165,165,251,229,253,133,251,176,  2,198,252,169
210 DATA 131,160,164,141,  2,  3,140,  3,  3, 76,118,165
220 IF S <> 15495 THEN PRINT "FEHLER IN DATAS !!" : END
230 PRINT "OK"
```

6.6 Die Benutzung von Token

Bisher waren wir es immer nur gewohnt, unsere neuen Assembler-Unterroutinen mit dem "SYS"-Befehl aufzurufen. Viel schöner wäre es jedoch, wenn wir statt des Befehls

```
SYS zn
```

dieselbe Routine mit einem richtigen Befehl, wie z.B.

```
AUTO zn
```

aufrufen könnten. Dies ist auch möglich. Das Betriebssystem bietet nämlich die Möglichkeit, einige, oft sehr wichtige BASIC- und Betriebssystem-Routinen, zu erweitern. Es wählt dabei folgenden Weg: Bei jedem Warmstart richtet sich das Betriebssystem einen bestimmten Speicherbereich im RAM ein, wo die sogenannten Sprungvektoren liegen. Sprungvektoren sind beim Commodore 64 zwei Bytes, die eine bestimmte Adresse enthalten. Nun gibt es die Möglichkeit in 6502-Assembler, indirekt zu einer Adresse zu springen. Das geschieht wie folgt:

```
JMP (vektor)
```

Hier wird nicht zu der Adresse "vektor" gesprungen, sondern zu der Adresse, die sich ergibt, wenn man den Inhalt von "vektor" und "vektor"+1 zusammensetzt. Als Beispiel nun zwei Routinen, die exakt das gleiche machen, nur, das die erste Routine sich der Vektor-Sprungmethode bedient:

```
Routine I                Routine II
-----------------------------------------
lda #$1e                      ;
sta $fa                  jmp $ab1e
lda #$ab                      ;
sta $fb
jmp ($00fa)
```

Genau diese Methode verwendet auch das Betriebssystem, bei einigen wichtigen Routinen. Dort zeigt der Vektor meistens auf den nachfolgenden Assemblerbefehl.

Die Tatsache jedoch, daß dieser Vektor im RAM liegt, können wir uns nun zu Nutze machen, indem wir diesen Zeiger einfach auf unsere eigene Routine "umbiegen" und nur, wenn es nötig ist, die alten Routinen aufrufen.

Im Folgenden finden Sie drei Routinen, mit denen Sie das BASIC um Befehle und Funktionen erweitern können, die mit Befehlsworten aufgerufen werden können. Kommen wir nun zur Funktion der einzelnen Routinen. Wie Sie vielleicht wissen, ersetzt das BASIC die Befehlsworte grundsätzlich durch ein Byte. Das ist platzsparend und im Ablauf später sehr viel flexibler und schneller. Diese Bytes werden "Token" genannt. Bei jeder Eingabe wandelt nun das Betriebssystem ein Befehlswort, das es erkannt hat, in ein solches Token um. Dabei wird natürlich berücksichtigt, ob dieses Befehlswort in einer DATA-Zeile oder in einer Gänsefüßchen-Zeile steht. Dafür ist die erste Routine gut. Die zweite Routine macht genau das Gegenteil. Sie wandelt nämlich das Token wieder in den Klartext um, wenn dies beim LISTen von Programmen benötigt wird. Dann gibt es noch eine dritte Routine, die ein Token als ein solches erkennt und die entsprechende Assemblerroutine anspringt. Die einzelnen genauen Abläufe der Routinen hier noch einmal zu beschreiben, wäre wenig sinnvoll, da wie ich meine die einzelnen Befehle genügend dokumentiert wurden. Sie selber müssen natürlich die einzelnen Vektoren auf die verschiedenen Routinen setzen. Dazu habe ich in der Zeile 140 einen Sprung zu dem Label "startprg" gesetzt, in das Sie Ihre Befehlserweiterung initialisieren können:

```
110:  c000                    .opt p1,oo
120:  c000                    *=   $c000
130:                  ;
140:  c000 4c 00 00           jmp  startprg ; erweiterung initialisieren
150:                  ;
160:                  ;konstanten
170:                  ;
190:  00d1            off     =    $d1
260:  0083            data    =    $83
270:  008f            rem     =    $8f
280:  0099            print   =    $99
```

```
290:  0008          char     =   8
300:  000b          count    =   11
310:  0071          pnt1     =   $71
320:  0022          quote    =   $22
330:  000f          flag     =   15
340:  007a          txtptr   =   $7a
350:  0200          buffer   =   $200
360:  a09e          table    =   $a09e
370:  0049          pnt2     =   $49
380:  ab47          charout  =   $ab47
390:  0308          cmdvek   =   $308
400:  030a          funvek   =   $30a
410:  000d          typflg   =   13
420:  0073          chrget   =   $73
430:  0079          chrgot   =   $79
440:  a7ed          execold  =   $a7ed
450:  a7ae          inter    =   $a7ae
460:  aef1          getterm  =   $aef1
470:  ad8d          checknum =   $ad8d
480:  0054          jump     =   $54
490:  ae8d          funktold =   $ae8d
500:  00cc          cmdstart =   $cc
510:  00fc          cmdend   =   $fc
520:                ;
530:  00fe          funstart =   $fe
540:  00fe          funend   =   $fe
550:                ;
560:  000d          typflag  =   13
570:                ;
580:                ;
590:                ; *** umwandlung in eigene tokens ***
600:                ;
610:  c003 a6 7a    owntok   ldx  txtptr  ; zeiger auf umzuwandelnde
                                            zeile
620:  c005 a0 04           ldy  #4       ; hochkommaflag loeschen
630:  c007 84 0f           sty  flag
640:  c009 bd 00 02 nextchar lda  buffer,x ; zeichen aus buffer holen
650:  c00c 10 07           bpl  normal   ; kein basictoken
660:  c00e c9 ff           cmp  #$ff     ; mit code fuer "pi"
                                            vergleichen
```

```
670:   c010 f0 3e          beq  takchar  ; wenn ja, dann so
                                           uebernehmen
680:   c012 e8             inx           ; sonst ignorieren
690:   c013 d0 f4          bne  nextchar  ; naechstes zeichen holen
700:                ;
710:   c015 c9 20   normal  cmp  #" "     ; space
720:   c017 f0 37          beq  takchar   ; so uebernehmen
730:   c019 85 08          sta  char      ; sonst zeichen
                                           zwischenspeichern
740:   c01b c9 22          cmp  #quote    ; gaensefuesschen ?
750:   c01d f0 55          beq  getchar   ; so uebernehmen
760:   c01f 24 0f          bit  flag      ; data-modus testen
770:   c021 70 2d          bvs  takchar   ; ja, dann so uebernehmen
780:   c023 c9 3f          cmp  #"?"      ; fragezeichen ?
790:   c025 d0 04          bne  skip
800:   c027 a9 99          lda  #print    ; durch print-token ersetzen
810:   c029 d0 25          bne  takchar   ; unbedingter sprung
820:   c02b c9 30   skip    cmp  #"0"      ; kleiner als "0"
830:   c02d 90 04          bcc  skip1     ; dann tabelle durchsuchen
840:   c02f c9 3c          cmp  #"<"      ; kleiner als "<"
850:   c031 90 1d          bcc  takchar   ; ja, dann so uebernehmen
860:   c033 84 71   skip1   sty  pnt1      ; zeiger merken
870:   c035 a0 00          ldy  #0        ; befehlswortzaehler auf
880:   c037 84 0b          sty  count     ; 0 setzen
890:   c039 88             dey           ;
900:   c03a 86 7a          stx  txtptr    ; zeiger merken
910:   c03c ca             dex
920:                ;
930:   c03d c8      cmploop iny
940:   c03e e8             inx
950:   c03f bd 00 02 testnext lda  buffer,x  ; zeichen im
960:   c042 38              sec           ; buffer mit befehlsworten in
der
970:   c043 f9 9e a0        sbc  table,y   ; tabelle vergleichen
980:   c046 f0 f5           beq  cmploop
990:   c048 c9 80           cmp  #$80      ; letzer buchstabe
                                           (geshifted)
1000:  c04a d0 2f           bne  nextcmd
1010:  c04c 05 0b           ora  count
```

```
1020:  c04e a4 71    takchar1 ldy  pnt1    ; zeiger wiederholen
1030:                 ;
1040:  c050 e8       takchar  inx
1050:  c051 c8                iny
1060:  c052 99 fb 01          sta  buffer-5,y  ; basiccode abspeichern
1070:  c055 c9 00             cmp  #0       ; wenn 0, dann zeilenende
1080:  c057 f0 38             beq  ende
1090:  c059 38                sec
1100:  c05a e9 3a             sbc  #":"
1110:  c05c f0 04             beq  skip2
1120:  c05e c9 49             cmp  #data-":"  ; mit data-code
                                                vergleichen
1130:  c060 d0 02             bne  skip3
1140:  c062 85 0f    skip2    sta  flag     ; data-flag setzen
1150:  c064 38       skip3    sec
1160:  c065 e9 55             sbc  #rem-":"  ; mit rem-code vergleichen
1170:  c067 d0 a0             bne  nextchar
1180:  c069 85 08             sta  char     ; zwischenspeichern
1190:  c06b bd 00 02 remloop  lda  buffer,x  ; zeichen bis
1200:  c06e f0 e0             beq  takchar   ; zum zeilenende
1210:  c070 c5 08             cmp  char     ; uebernehmen
1220:  c072 f0 dc             beq  takchar
1230:  c074 c8       getchar  iny
1240:  c075 99 fb 01          sta  buffer-5,y
1250:  c078 e8                inx
1260:  c079 d0 f0             bne  remloop
1270:                 ;
1280:  c07b a6 7a    nextcmd  ldx  txtptr
1290:  c07d e6 0b             inc  count    ; tokenzaehler erhoehen
1300:  c07f c8       weiter   iny          ;
1310:  c080 b9 9d a0          lda  table-1,y
1320:  c083 10 fa             bpl  weiter   ; wortende suchen
1330:  c085 b9 9e a0          lda  table,y  ; buffer noch nicht zu ende
1340:  c088 d0 b5             bne  testnext  ; dann weiter suchen
1350:  c08a f0 0f             beq  newtok   ; neue
                                              Tokenumwandlungsroutine
1360:                 ;
1370:  c08c bd 00 02 notfound lda  buffer,x  ; wort nicht gefunden
1380:  c08f 10 bd             bpl  takchar1
1390:                 ;
```

```
1400:  c091 99 fd 01 ende     sta  buffer-3,y
1410:                ;
1420:  c094 c6 7b            dec  txtptr+1
1430:  c096 a9 ff            lda  #$ff
1440:  c098 85 7a            sta  txtptr
1450:  c09a 60               rts
1460:                ;
1470:                ;*** verarbeitung ***
1480:                ;
1490:  c09b a0 00    newtok   ldy  #0
1500:  c09d b9 53 c1         lda  newtab,y
1510:  c0a0 d0 02            bne  newtest
1520:  c0a2 c8       newcmp   iny
1530:  c0a3 e8               inx
1540:  c0a4 bd 00 02 newtest  lda  buffer,x  ; zeichen im
1550:  c0a7 38               sec             ; buffer mit neuen befehls-
1560:  c0a8 f9 53 c1         sbc  newtab,y   ; worten vergleichen
1570:  c0ab f0 f5            beq  newcmp
1580:  c0ad c9 80            cmp  #$80       ; wort zu ende
1590:  c0af d0 04            bne  nextnew
1600:  c0b1 05 0b            ora  count
1610:  c0b3 d0 99            bne  takchar1
1620:                ;
1630:  c0b5 a6 7a    nextnew  ldx  txtptr
1640:  c0b7 e6 0b            inc  count      ; tokenzaehler erhoehen
1650:  c0b9 c8       weiter1  iny
1660:  c0ba b9 52 c1         lda  newtab-1,y
1670:  c0bd 10 fa            bpl  weiter1    ; ende des wortes suchen
1680:  c0bf b9 53 c1         lda  newtab,y   ; zeichen holen
1690:  c0c2 d0 e0            bne  newtest
1700:  c0c4 f0 c6            beq  notfound   ; ende, dann wort nicht
                                                gefunden
1710:                ;
1720:                ;
1730:                ;**** neue list-routine ****
1740:                ;
1750:                ;
1760:  c0c6 10 0f    neulist  bpl  out        ; kein interpretercode, so
                                                ausgeben
1770:  c0c8 24 0f            bit  flag       ; hochkommamodus ?
```

```
1780:  c0ca 30 0b           bmi  out      ; so ausgeben
1790:  c0cc c9 ff           cmp  #$ff     ; code fuer "pi" ?
1800:  c0ce f0 07           beq  out      ; so ausgeben
1810:  c0d0 c9 cc           cmp  #$cc     ; groesser als groesstes
                                            altes token
1820:  c0d2 b0 06           bcs  newlist  ; ja, dann neue listroutine
1830:                ;
1840:  c0d4 4c 24 a7        jmp  $a724    ; sonst zur alten routine
1850:  c0d7 4c f3 a6 out    jmp  $a6f3    ; zeichen ausgeben
1860:                ;
1870:  c0da 38       newlist sec
1880:  c0db e9 cb           sbc  #$cb
1890:  c0dd aa              tax
1900:  c0de 84 49           sty  pnt2     ; das dem token
1910:  c0e0 a0 ff           ldy  #-1      ; entsprechende befehls-
1920:  c0e2 ca       next   dex           ; wort suchen
1930:  c0e3 f0 08           beq  found
1940:  c0e5 c8       loop1  iny
1950:  c0e6 b9 53 c1        lda  newtab,y
1960:  c0e9 10 fa           bpl  loop1
1970:  c0eb 30 f5           bmi  next
1980:                ;
1990:  c0ed c8       found  iny           ; und ausgeben
2000:  c0ee b9 53 c1        lda  newtab,y
2010:  c0f1 30 05           bmi  oldend
2020:  c0f3 20 47 ab        jsr  charout
2030:  c0f6 d0 f5           bne  found
2040:                ;
2050:  c0f8 4c ef a6 oldend jmp  $a6ef    ; zurueck zur alten routine
2060:                ;
2070:                ;
2080:                ;**** einbindung neuer tokens ****
2090:                ;
2100:                ;
2110:  c0fb 20 73 00 neutok jsr  chrget   ; zeichen holen
2120:  c0fe 20 04 c1        jsr  testcmd  ; befehl (alt und neu)
                                            ausfuehren
2130:  c101 4c ae a7        jmp  inter    ; zur interpreterschleife
2140:                ;
2150:  c104 c9 cc    testcmd cmp #cmdstart
```

```
2160:  c106 90 04           bcc  oldcmd
2170:  c108 c9 fd           cmp  #cmdend+1
2180:  c10a 90 06           bcc  oknew   ; neuen befehl erkannt
2190:  c10c 20 79 00 oldcmd jsr  chrgot
2200:  c10f 4c ed a7        jmp  execold  ; alte befehle ausfuehren
2210:               ;
2220:  c112 38      oknew   sec          ; adresse des neuen
2230:  c113 e9 cc           sbc  #cmdstart  ; befehls errechnen
2240:  c115 0a              asl
2250:  c116 aa              tax
2260:  c117 bd 5b c1        lda  cmdtab+1,x
2270:  c11a 48              pha          ; und auf stack speichern
2280:  c11b bd 5a c1        lda  cmdtab,x
2290:  c11e 48              pha
2300:  c11f 4c 73 00        jmp  chrget  ; befehl ausfuehren
2310:               ;
2320:  c122 a9 00   newfun  lda  #0
2330:  c124 85 0d           sta  typflag
2340:  c126 20 73 00        jsr  chrget  ; befehlstoken holen
2350:  c129 c9 fe           cmp  #funstart  ; mit funstart vergleichen
2360:  c12b 90 04           bcc  oldfun  ; alte funktion
2370:  c12d c9 ff           cmp  #funend+1
2380:  c12f 90 06           bcc  ok1new  ; neue funktion
2390:  c131 20 79 00 oldfun jsr  chrgot
2400:  c134 4c 8d ae        jmp  funktold
2410:               ;
2420:  c137 38      ok1new  sec
2420:  c138 e9 fe           sbc  #funstart
2430:  c13a 0a              asl
2440:  c13b 48              pha
2450:  c13c 20 73 00        jsr  chrget  ; zeichen holen
2460:  c13f 20 f1 ae        jsr  getterm  ; term in klammern holen
2470:  c142 aa              tax
2480:  c143 bd 5c c1        lda  funtab,x  ; adresse in vektor
                                           speichern
2490:  c146 85 55           sta  jump+1
2500:  c148 bd 5d c1        lda  funtab+1,x
2510:  c14b 85 56           sta  jump+2
2520:  c14d 20 54 00        jsr  jump    ; funktion aufrufen
```

```
2530:  c150 4c 8d ad          jmp  checknum  ; und auf nummerisch
                                               pruefen
2540:                ;
2550:                ;*** befehlsworte ***
2560:                ;
2570:  c153 44 49 53 newtab   .asc "displa" ; "DISPLAY" als Beispiel
2570:  c159 d9                .byt "y"+$80
2580:                ;
2590:                ;
2600:                ;*** cmdtab ***
2610:                ;
2620:  c15a           cmdtab   .wor display
2630:                ;
2640:                ;*** funtab ***
2650:                ;
2660:  c15a 00 00     funtab   .wor 0
```

Folgende Vektoren müssen umgebogen werden:

$304/$305 - Umwandlung einer Zeile in Token
$306/$307 - Umwandlung von Token in den Klartext (LIST)
$308/$309 - Ausführung der neuen Befehle
$30A/$30B - Ausführung der neuen Funktionen

Die einzelnen Routinen tragen in diesem Falle folgenden Namen:

```
owntok
neulist
neutok
newfun
```

Weiterhin ist zu beachten, daß es für die Worte der Funktionen keine eigenen Wort-Tabelle gibt. Diese Funktionsworte müssen direkt hinter den Befehlsworten stehen.

7. Kapitel

Schnittstellen

7.1 Der User-Port

Wie Sie vielleicht wissen, verfügt der Commodore 64 über eine Schnittstelle, die vom Betriebssystem zwar nicht benutzt wird, die Sie jedoch für Ihre eigenen Zwecke nutzen können. Gesteuert wird diese Schnittstelle über einen speziellen Baustein, den CIA 6526. Dies hat nichts mit dem amerikanischen Geheimdienst zu tun, sondern ist vielmehr die Abkürzung von Complex Interface Adapter. Von diesem Baustein sind nun zwei in Ihrem Commodore 64 vorhanden, wobei einer davon, wie schon gesagt, unter anderem auch für den User-Port zuständig ist. Im folgenden will ich Ihnen nun einmal die Register des 6526 beschreiben, damit Sie anhand dieser den User-Port für Ihre eigenen Anwendungen nutzen können:

7.2 Registerbeschreibung des 6526

Reg 0 *PRA (Port Register A)*
Zugriff: Read/Write
Bits 0-7: Dieses Register entspricht dem Zustand der Pins PA0-7.

Reg 1 *PRB (Port Register B)*
Zugriff: Read/Write
Bits 0-7: Dieses Register entspricht dem Zustand der Pins PB0-7

Reg 2 *DDRA (Datenrichtung Register A)*
Zugriff: Read/Write
Bits 0-7: Diese Bits bestimmen die Datenrichtung (0=Eingabe; 1= Ausgabe) der korrespondierenden Datenbits des Port A.

Reg 3 *DDRB (Datenrichtung Register B)*
Zugriff: Read/Write
Bits 0-7: Diese Bits bestimmen die Datenrichtung der entsprechenden Datenbits des Ports B.

Reg 4 *TA LO (Timer A, Low-Byte)*
Zugriff: Read
Bits 0-7: Dieses Register gibt den augenblicklichen Zustand des niederwertigen Bytes von Timer A wieder.
Zugriff: Write
Bits 0-7: In dieses Register wird das niederwertige Byte des Wertes geladen, von dem der Timer bis auf Null zählen soll.

Reg 5 *TA HI (Timer A,High-Byte)*
Zugriff: Read
Bits 0-7: Dieses Register gibt den augenblicklichen Zustand des höherwertigen Bytes von Timer A wieder.
Zugriff: Write
Bits 0-7: In dieses Register wird das höherwertige Byte des Wertes geladen, von dem der Timer bis auf Null zählen soll.

Reg 6 *TB LO (Timer B, Low-Byte)*
Siehe Register 4

Reg 7 *TB HI (Timer B, High-Byte)*

Reg 8 *TOD 10THS (Uhr 1/10 Sekunden)*
Zugriff: Read
Bits 0-3: Zehntelsekunden im BCD-Format
Bits 4-7: Immer Null
Zugriff: Write und CRB Bit 7=0
Bits 0-3: Zehntelsekunden im BCD-Format
Bits 4-7: Müssen Null sein!
Zugriff: Write und CRB Bit 7=1
Bits 0-3: Vorwahl der Zehntelsekunden der Alarmzeit im BCD-Format
Bits 0-4: Müssen Null sein.

Reg 9 *TOD SEC (Uhr Sekunden)*
Zugriff: Read
Bits 0-3: Einersekunden im BCD-Format
Bits 4-6: Zehnersekunden im BCD-Format

Bit 7: Immer Null
Weitere Zugriffsarten analog zu Register 8.

Reg 10 *TOD MIN (Uhr Minuten)*
Zugriff: Read
Bits 0-3: Einerminuten im BCD-Format
Bits 4-6: Zehnerminuten im BCD-Format
Bit 7: Immer Null
Weitere Zugriffsarten analog zu Register 8.

Reg 11 *TOD HR (Uhr Stunden)*
Zugriff: Read
Bits 0-3: Einerstunden im BCD-Format
Bit 4: Zehnerstunden der Uhr
Bits 5-6: Immer Null
Bit 7: 0=Vormittag, 1=Nachmittag
Weitere Zugriffsarten analog zu Register 8

Reg 12 *SDR (Serial Data Register)*
Zugriff: Read/Write
Bits 0-7: Aus diesem Register werden die Daten bitweise zum Pin SP hinausgeschoben bzw. vom Pin SP in dieses Register hineingeschoben.

Reg 13 *ICR (Interrrupt Control Register)*
Zugriff: Read (INT DATA):

Achtung: Beim Lesen dieses Registers werden alle Bits gelöscht.

Bit 0: 1=Unterlauf Timer A
Bit 1: 1=Unterlauf Timer B
Bit 2: 1=Gleichheit von Uhrzeit und gewählter Alarmzeit
Bit 3: 1=SDR voll/leer (abhängig von der Betriebsart)
Bit 4: 1=Signal am Pin FLAG aufgetreten
Bit 5-6: Immer Null
Bit 7: Übereinstimmung mindestens eines Bits von INT MASK und INT DATA aufgetreten.

Zugriff: Write (INT MASK)
Bit 7: 1=Jedes 1-Bit setzt das korespondierende Maskenbit. Die anderen bleiben unberührt. 0=Jedes 1-Bit löscht das korrespondierende Maskenbit. Die anderen Bits bleiben unberührt.

Reg 14 *CRA (Control Register A)*
Zugriff: Read/Write
Bit 0: Timer A 1=Start; 0=Stop
Bit 1: 1=Unterlauf von Timer A wird an Pin PB6 signalisiert.
Bit 2: 1=Jeder Unterlauf von Timer A kippt PB6 in die jeweils andere Lage. 0=Jeder Unterlauf von Timer A erzeugt an PB6 einen HI-Impuls mit der Länge eines Systemtaktes
Bit 3: 1=Timer A zählt nur einmal vom Ausgangswert auf Null und hält dann an 0=Timer A zählt fortlaufend vom Ausgangswert auf Null
Bit 4: 1=Unbedingtes Laden eines neuen Startwertes in Timer A. Dieses Bit fungiert als Strobe. Es muß bei jedem unbedingten Laden neu gesetzt werden.
Bit 5: Dieses Bit bestimmt die Quelle des Timertriggers. 1=Timer zählt steigende CNT-Flaken, 0=Timer zählt Systemtaktpulse
Bit 6: 1=SP ist Ausgang; 0=SP ist Eingang
Bit 7: 1=Echtzeituhr-Trigger beträgt 50HZ
0=Echtzeituhr-Trigger beträgt 60HZ

Reg 15 *CBR (Control Register B)*
Bits 0-4: Diese Bits haben die gleiche Bedeutung wie in Register 14, allerdings bezogen auf Timer B und PB7
Bits 5-6: Diese Bits bestimmen die Quelle des Triggers für Timer B. 00=Timer zählt Systemtakt; 01=Timer zählt steigende CNT-Flaken, 10=Timer zählt Unterläufe von Timer A, 11=Timer zählt Unterläufe von Timer a, wenn CNT=1 ist.
Bit 7: 1=Alarm setzen; 0=Uhrzeit setzen.

7.3 Eine sinnvolle Anwendung des USER-Ports: Schnittstelle für einen Centronics-Drucker

Der Commodore 64 hat eine Schnittstelle, die normalerweise vom Betriebssystem nicht benutzt wird, und die Ihnen für eigene Anwendungen zur Verfügung steht. Diese Schnittstelle besteht aus einem 8-Bit-Port und zwei Handshake-Leitungen. Der 8-Bit-Port kann sowohl auf Ausgabe als auch auf Eingabe geschaltet sein und zwar unabhängig für jedes Bit.

Diese Schnittstelle eignet sich hervorragend zur Realisierung einer Druckerschnittstelle. Da die meisten Drucker serienmäßig über eine Centronics-Schnittstelle verfügen (z.B. Epson-Drucker, Centronics-Drucker), wollen wir diese Schnittstelle verwirklichen. Zudem läßt sich diese Schnittstelle besonders einfach handhaben. Hier kurz das Verfahren:

Die 8 Bit eines Byte werden gleichzeitig parallel über acht Datenleitungen übertragen. Damit bei der Übertragung keine Daten verloren gehen, werden zwei sogenannte 'Handshake-Leitungen' benutzt. Ehe der Rechner ein Datenbyte an den Drucker schickt, prüft er ob der Drucker bereit ist, Daten zu empfangen. Dies geschieht über die BUSY-Leitung. Ist die BUSY-Leitung high, ist der Drucker beschäftigt und der Rechner muß warten. Wenn der Drucker bereit ist, legt der Rechner die Daten auf den Port und signalisiert dies dem Drucker über die STROBE-Leitung. Der Drucker übernimmt die Daten und setzt die BUSY-Leitung solange high, bis die Daten verarbeitet (gespeichert oder gedruckt) sind. Jetzt kann das nächste Datenbyte übertragen werden. Durch diese Prozedur wird sichergestellt, daß der Drucker auch jedes Datenbyte erhält, das der Rechner abschickt.

Nun zur Programmierung der Schnittstelle. Damit die Daten mit PRINT# zum Drucker geschickt werden können, muß die Software ins Betriebssystem eingebunden werden. Zusätzlich ergibt sich jedoch noch ein Problem:

Die meisten Drucker mit Centronics-Schnittstelle besitzen einen ASCII-Zeichensatz, der sich vom Zeichensatz des Commodore 64 unterscheidet. Wir müssen deshalb eine Codewandlung vornehmen, damit der Drucker das ASCII-Zeichen erhält, das dem Zeichen des Commodore 64 entspricht. Diese Wandlung ist so ausgelegt, daß die Zeichen, wie sie im Groß/Kleinschrift-Modus auf dem Bildschirm erscheinen, ebenso auf dem Drucker wiedergegeben werden. Zusätzlich ist es jedoch manchmal erforderlich, die Zeichen so zu übertragen, wie sie der Rechner sendet. Dies ist dann nötig, wenn z.B. Daten für Einzelpunktgrafik übertragen werden. In dem untenstehenden Programm wurde dies so gelöst:

Gibt man beim OPEN-Befehl keine Sekundäradresse an, so werden die Daten umgewandelt, damit die Groß/Kleinschrift richtig erscheint. Mit einer Sekundäradresse von eins werden die Daten ohne Wandlung übertragen. Als Geräteadresse wurde 2 gewählt, was normalerweise die RS-232-Schnittstelle ist. Da diese Schnittstelle jedoch nicht zusammen mit unserer betrieben werden kann, gibt es deshalb keine Probleme.

Wollen Sie z.B. ein Programmlisting über diese Schnittstelle ausgeben, sehen die Befehle dazu so aus:

```
OPEN 1,2 : CMD 1 : LIST
```

Nach dem der Cursor wieder erscheint, geben Sie

```
PRINT#1 : CLOSE 1
```

ein, um den CMD-Modus wieder aufzuheben und den Kanal zu schließen. Wollen Sie Grafikdaten übertragen, sieht die Programmierung so aus:

```
OPEN 1,2,1
PRINT# 1, ....
CLOSE 1
```

Zum hardwaremäßigen Anschluß brauchen wir lediglich ein Kabel mit USER-Port-Stecker auf der einen Seite und einem Centronics-Stecker auf der anderen. Die Pinbelegung des Kabels ist am Ende des Assemblerlistings angegeben. Beim Anschluß des Druckers gehen Sie so vor, daß Sie zuerst den Commodore 64 über das Kabel mit dem Drucker verbinden, dann den Rechner und danach den Drucker einschalten. Laden Sie jetzt das Maschinenprogramm und initialisieren Sie es mit SYS 12*4096.

```
; Centronics-Schnittstelle für CBM 64
; Anschluß an USER-Port
;
; Definition der I/O-Vektoren
;
200:  031a                  *=   $31a
210:  031c          openv   *=   *+2
220:  031e          closev  *=   *+2
230:  0320          chkinv  *=   *+2
240:  0322          chkoutv *=   *+2
260:  0326                  *=   *+4
270:  0328          bsoutv  *=   *+2
280:  0328          xreg    =    $97     ; Speicher für Register
;
; Definition der Ports
;
284:  dd00                  *=   $dd00   ; CIA2
285:  dd01          porta   *=   *+1     ; Port
286:  dd02          portb   *=   *+1
```

```
287:  dd03           drra      *=   *+1      ; Datenrichtung
288:  dd0d           drrb      *=   *+10
289:  dd0e           icr       *=   *+1      ; Interrupt Control Register
290:  dd0e           lf        =    $b8
291:  dd0e           sa        =    $b9
292:  dd0e           fa        =    $ba
293:  dd0e           nmbfiles  =    $98
294:  dd0e           lftab     =    $259
295:  dd0e           fatab     =    $263
296:  dd0e           satab     =    $26d
297:  dd0e           srchfil   =    $f30f
                     ;
                     ; Initialisierung
                     ;
330:  c000           init      *=   $c000
340:  c000 a9 58               lda  #< open
350:  c002 a0 c0               ldy  #> open
360:  c004 8d 1a 03            sta  openv
370:  c007 8c 1b 03            sty  openv+1
380:  c00a a9 8b               lda  #< close
390:  c00c a0 c0               ldy  #> close
400:  c00e 8d 1c 03            sta  closev
410:  c011 8c 1d 03            sty  closev+1
420:  c014 a9 a3               lda  #< chkin
430:  c016 a0 c0               ldy  #> chkin
440:  c018 8d 1e 03            sta  chkinv
450:  c01b 8c 1f 03            sty  chkinv+1
460:  c01e a9 ba               lda  #< chkout
470:  c020 a0 c0               ldy  #> chkout
480:  c022 8d 20 03            sta  chkoutv
490:  c025 8c 21 03            sty  chkoutv+1
540:  c028 a9 d1               lda  #< bsout
550:  c02a a0 c0               ldy  #> bsout
560:  c02c 8d 26 03            sta  bsoutv
570:  c02f 8c 27 03            sty  bsoutv+1
580:  c032 a9 ff               lda  #$ff
590:  c034 8d 03 dd            sta  drrb     ; Port B auf Ausgabe
600:  c037 ad 00 dd            lda  porta
```

```
610:   c03a 09 04              ora  #%100   ; Strobe hi
610:   c03c 8d 00 dd           sta  porta
620:   c03f 60                 rts
;
; Ausgabe mit Handshake
; Daten auf Port B
; Strobe auf PA2
; BUSY über FLAG nach ICR
;
795:   c040 8d 01 dd ausgabe   sta  portb   ; Daten ausgeben
800:   c043 a9 10              lda  #%10000  ; Maske für 'FLAG'-Bit
810:   c045 2c 0d dd testbusy  bit  icr
820:   c048 f0 fb              beq  testbusy
830:   c04a ad 00 dd           lda  porta
840:   c04d 29 fb              and  #%11111011 ; Strobe lo
850:   c04f 8d 00 dd           sta  porta
860:   c052 09 04              ora  #%100     ;  Strobe hi
870:   c054 8d 00 dd           sta  porta
880:   c057 60                 rts
;
900:   c058 a6 b8    open      ldx  lf      ; logische Filenummer
910:   c05a f0 05              beq  openerr
930:   c05c 20 0f f3           jsr  srchfil  ; sucht Filenummer
940:   c05f d0 03              bne  *+5
950:   c061 4c fe f6 openerr   jmp  $f6fe   ; 'file open error'
960:   c064 a6 98              ldx  nmbfiles. ; Anzahl der offenen Files
970:   c066 e0 0a              cpx  #10
980:   c068 90 03              bcc  *+5
990:   c06a 4c fb f6           jmp  $f6fb   ; 'too many files error'
1000:  c06d e6 98              inc  nmbfiles
1010:  c06f a5 b8              lda  lf
1020:  c071 9d 59 02           sta  lftab,x
1030:  c074 a5 b9              lda  sa
1040:  c076 09 60              ora  #$60
1050:  c078 9d 6d 02           sta  satab,x
1060:  c07b a5 ba              lda  fa
1070:  c07d 9d 63 02           sta  fatab,x
1080:  c080 c9 02              cmp  #2
1090:  c082 d0 02              bne  *+4
1100:  c084 18                 clc
```

```
1100:  c085 60              rts          ; fertig
1110:  c086 c9 00           cmp  #0
1120:  c088 4c 77 f3        jmp  $f377   ; weiter wie gehabt
;
1300:  c08b 20 14 f3 close  jsr  $f314   ; sucht logische Filenummer
1310:  c08e f0 02           beq  *+4
1320:  c090 18              clc
1320:  c091 60              rts          ; fertig
1330:  c092 20 1f f3        jsr  $f31f   ; setzt File Parameter
1340:  c095 8a              txa
1340:  c096 48              pha
1350:  c097 a5 ba           lda  fa
1360:  c099 c9 02           cmp  #2
1370:  c09b f0 03           beq  *+5
1380:  c09d 4c 9d f2        jmp  $f29d   ; normal weiter
1390:  c0a0 4c f1 f2        jmp  $f2f1   ; Eintrag in Tabelle löschen
;
1400:  c0a3 20 0f f3 chkin  jsr  srchfil  ; sucht Filenummer
1410:  c0a6 f0 03           beq  *+5
1420:  c0a8 4c 01 f7        jmp  $f701   ; 'file not open error'
1430:  c0ab 20 1f f3        jsr  $f31f   ; setzt Fileparameter
1440:  c0ae a5 ba           lda  fa
1450:  c0b0 c9 02           cmp  #2
1460:  c0b2 d0 03           bne  *+5
1470:  c0b4 4c 0a f7        jmp  $f70a   ; 'not input file error'
1480:  c0b7 4c 19 f2        jmp  $f219
;
1500:  c0ba 20 0f f3 chkout jsr  srchfil  ; sucht Filenummer
1510:  c0bd f0 03           beq  *+5
1520:  c0bf 4c 01 f7        jmp  $f701   ; 'file not open error'
1530:  c0c2 20 1f f3        jsr  $f31f   ; setzt Fileparameter
1540:  c0c5 a5 ba           lda  fa
1550:  c0c7 c9 02           cmp  #2
1560:  c0c9 d0 03           bne  *+5
1570:  c0cb 4c 75 f2        jmp  $f275
1580:  c0ce 4c 5b f2        jmp  $f25b
;
1600:  c0d1 48       bsout  pha
1610:  c0d2 a5 9a           lda  $9a     ; Outputgerät
1620:  c0d4 c9 02           cmp  #2
```

```
1630:  c0d6 f0 03            beq  *+5
1640:  c0d8 4c cd f1         jmp  $f1cd   ; normal weiter
1641:  c0db a5 b9            lda  sa      ; Sekundäradresse
1641:  c0dd 29 0f            and  #%1111
1642:  c0df d0 0a            bne  out     ; ungleich null
1642:  c0e1 86 97            stx  xreg
1643:  c0e3 68               pla
1643:  c0e4 aa               tax
1643:  c0e5 bd f3 c0         lda  table,x ; Code aus Tabelle holen
1643:  c0e8 a6 97            ldx  xreg
1643:  c0ea 24               .byt $24
1650:  c0eb 68         out   pla
1655:  c0ec 48               pha
1660:  c0ed 20 40 c0         jsr  ausgabe ; Zeichen ausgeben
1660:  c0f0 68               pla
1660:  c0f1 18               clc
1660:  c0f2 60               rts
1665:  c0f3            table =  *

c0f3 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
c103 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
c113 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
c123 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
c133 40 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
c143 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 5f
c153 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
c163 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f
c173 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
c183 90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f
c193 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af
c1a3 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
c1b3 c0 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
c1c3 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d de df
c1d3 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
c1e3 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
```

USER PORT	-	CENTRONICS
A	GND	16
B	FLAG - BUSY	11
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	PA2 - STROBE	1

7.4 Datenübertragung von und zu anderen Rechnern

Nehmen wir einmal folgenden Fall an:

Sie besitzen außer Ihrem C64, den Sie sich wegen der Möglichkeiten in Farbe und Ton zugelegt haben, noch einen CBM-8032. Wie schön wäre es doch, wenn Sie Daten, die aus dem 8032 stammen, zwecks farbiger Anzeige auf den CBM64 transferieren könnten, und zwar in dem Augenblick, wo sie auch anfallen.

Sie könnten wohl die Daten mittels Kassette austauschen, falls Sie ein solches Gerät besitzen, aber das ist ein umständliches und langwieriges Unterfangen.

Um Ihnen den Nutzen des User-Port einmal aufzuzeigen (die wenigsten wissen damit etwas anzufangen), haben wir eben dieses Beispiel gewählt und dazu auch gleich ein kleines Programm geschrieben, das es erlaubt, Daten von und zu Ihrem C64 zu übertragen. In unserem Falle ist der Partner ein CBM8032. Es ist jedoch durchaus denkbar, dieses Verfahren auch mit anderen Rechnern zu versuchen, die über eine ähnliche Einrichtung wie den User-Port verfügen.

Einzige Voraussetzung ist eben, daß diese Einrichtung über acht Datenleitungen und zwei Handshakeleitungen verfügt. Geeignet wäre z.B. auch eine Centronicsschnittstelle, über die ja auch im Abschnitt 7.1 einiges gesagt ist.

Das Programm, von dem im folgenden die Rede sein wird, benötigt natürlich zum geordneten Ablauf eine Verbindung zwischen den beiden Rechnern.

Anweisungen, wie eine solche Leitung beschaffen sein muß, finden Sie hinter dem Programmlisting.

Der Einsatz dieser Routinen ist sehr einfach.

Nachdem Sie ein entsprechendes Kabel angefertigt und die in Frage kommenden Rechner damit verbunden haben, kann es eigentlich schon losgehen.

Das beschriebene Programm besteht aus zwei Unterprogrammen: Eines zum Senden und eines zum Empfang.

Diese Unterprogramme sollten Sie in Ihr eigenes Programm einbinden. Wollen Sie Zeichen senden, so schaffen Sie diese nach D$ und sagen GOSUB1000.

Wollen Sie empfangen, so sagen sie GOSUB2000 und haben nach der Rückkehr die Zeichen in D$ zur Verfügung, so einfach ist das.

Nachfolgend finden Sie zwei Programmlistings: eines für den C64 und eins für den CBM8032. In der Struktur sind beide gleich, jedoch sind die User-Port-Adressen zwangsläufig andere.

Außerdem unterscheiden sie sich noch in der Zeile 2010. Hier muß mit Rücksicht auf den andersartigen Baustein, der im C-64 den User-Port handhabt, ein Sprung eingefügt werden.

```
995 rem subroutinensammlung zur uebertragung
    von daten ueber den user-port
996 rem version für 6522 an adresse 59456 des CBM8032
998 rem
999 rem senden eines strings
1000 poke 59459,255:rem datenrichtung ausgang
1005 poke 59468,peek(59468) or 224:rem cb2 high
1010 for i=1 to len(d$):rem sendeschleife für d$
1020 x=asc(mid$(d$,i,1)):poke 59457,x:rem daten ausgeben
1030 poke 59468,peek(59468) and 223:rem cb2 low
1040 wait 59469,2:
     rem warten bis daten empfangen
1050 poke 59468,peek(59468) or 224:rem cb2 high
1060 next
1070 poke 59457,0:poke 59459,0:rem port rücksetzen
1080 return
1996 rem
1997 rem empfangen von d$
1998 rem
1999 rem
2000 wait 59469,2:
     rem anfang der übertragung abwarten
2010 d$="":rem d$ vorbereiten
2020 ti$="000000"
2030 if ti>120 then 2090
2040 if (peek(59469) and 2)=0 then 2030:
     rem warten auf datenbyte
2050 x=peek(59457):d$=d$+chr$(x)
2060 poke 59468,peek(59468) and 223:rem cb2 low
2070 poke 59468,peek(59468) or 224:
     rem cb2 high = empfangsbestätigung
2080 goto 2020
2090 return
```

```
995 rem subroutinensammlung zur Übertragung
    von daten über den user-port
996 rem version fuer 6526 an adresse 56576 des CBM64
998 rem
999 rem senden eines strings
1000 poke 56579,255:rem datenrichtung ausgang
1010 for i=1 to len(d$):rem sendeschleife für d$
1020 x=asc(mid$(d$,i,1)):poke 56577,x:
     rem daten ausgeben
1030 poke 56576,147:rem pa2 low
1040 wait 56589,16:
     rem warten bis daten empfangen
1050 poke 56576,151:rem pa2 high
1060 next
1070 poke 56577,0:poke 56579,0:rem port rücksetzen
1080 return
1996 rem
1997 rem empfangen von d$
1998 rem
1999 rem
2000 wait 56589,16:
     rem anfang der übertragung abwarten
2010 d$="":goto 2050:rem d$ vorbereiten
2020 ti$="000000"
2030 if ti>120 then 2090
2040 if (peek(56589) and 16)=0 then 2030:
     rem warten auf datenbyte
2050 x=peek(56577):d$=d$+chr$(x)
2060 poke 56576,147:rem pa2 low
2070 poke 56576,151:rem pa2 high:
     rem empfangsbestätigung
2080 goto 2020
2090 return
```

Ein kleines Beispiel soll die Anwendung der Routinen verdeutlichen:

Vorausgesetzt, sie haben in beiden Rechnern die entsprechende Routine geladen, fügen Sie nur noch im Sender die Zeile

```
10 d$="test":gosub1000:end
```

ein, und im Empfänger die Zeile

```
10 gosub2000:print d$:end
```

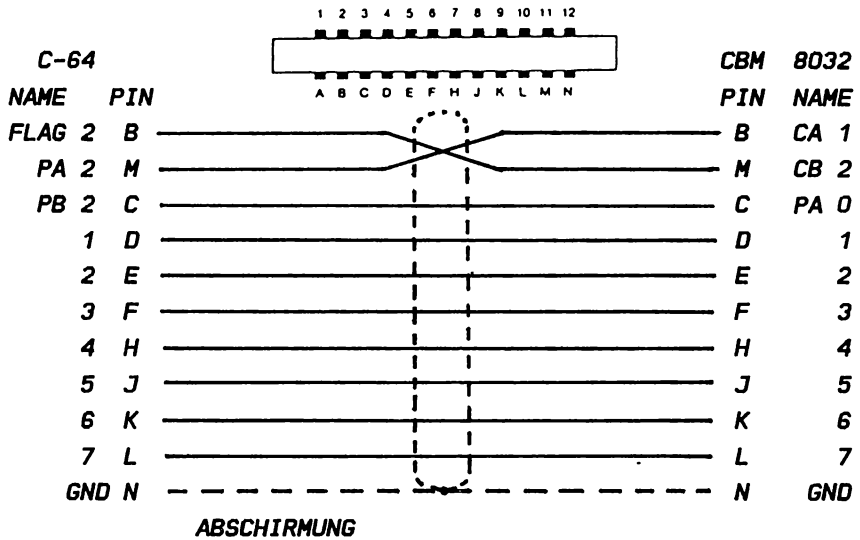
Starten Sie nun beide Programme und schauen Sie, was geschieht.

Die Verbindungsleitung ist nach untenstehendem Diagramm anzufertigen.

Wir empfehlen als Kabel eine 10-adrige, abgeschirmte Leitung mit einem Querschnitt der Einzeladern von 0,14-0,35 mm^2
Die Abschirmung ist auf beiden Seiten mit dem Pin GND zu verbinden.

Bei unseren Versuchen funktionierte die Anordnung bei einer Leitungslänge von 5m fehlerfrei. Ob wesentlich größere Längen noch möglich sind, kann nicht vorhergesagt werden. Das hängt u.a. auch von der Umgebung und deren Störfeldern ab. Zum Beispiel erzeugen Elektromotoren (Waschmaschine, Staubsauger) starke Störungen, die zu Fehlern in der Übertragung führen können.

Als völlig unbedenklich kann eine Leitungslänge von 3m angesehen werden.


1 2 3 4 5 6 7 8 9 10 11 12
A B C D E F H J K L M N
C-64
NAME PIN
FLAG 2 B
PA 2 M
PB 2 C
1 D
2 E
3 F
4 H
5 J
6 K
7 L
GND N
CBM 8032
PIN NAME
B CA 1
M CB 2
C PA 0
D 1
E 2
F 3
H 4
J 5
K 6
L 7
N GND
ABSCHIRMUNG

8. Kapitel

Kurz-Tips & Tricks

Wo sieht man sie nicht Heutzutage? In fast jeder Computer-Zeitschrift findet man, die kurzen Tips & Tricks. Meistens mit dem Wortlaut:

"Durch POKE soundso, sowieso erreichen sie soetwas"

Doch warum das Ganze so ist, wird dort nicht erklärt. Diesem will ich nun mit dem Folgenden abhelfen. Die wichtigsten Tips & Tricks für den Commodore 64 sind hier zusammengestellt, und, ausführlich erklärt. Vielleicht kennen Sie sogar den einen oder anderen Trick oder Programmierkniff noch nicht und können ihn, wenn er Ihnen nützlich erscheint, in Ihre eigenen Programme einbauen. Fangen wir jedoch nun mit den Erklärungen der Tips & Tricks an:

8.1 Directory ohne Programmverlust

Durch einen ganz einfachen Trick, haben Sie die Möglichkeit das Inhaltsverzeichnis einer Diskette aufzulisten, ohne dabei Ihr momentanes Basicprogramm zu zerstören. Durch die Zeichenfolge:

```
POKE 44,PEEK(46)+1
```

Setzen Sie den Basicanfang auf das freie Ram hinter dem Basicprogramm. Danch können Sie Ihr Inhaltsverzeichnis wie gewohnt einladen und auf Drucker und Bildschirm ausgeben. Durch ein anschließendes

```
POKE 44,8
```

setzen Sie dann den Basicprogramm-Zeiger wieder auf den alten

Wert zurück. Bitte beachten Sie, daß alle Variablen bei einer solchen Operation gelöscht werden.

8.2 Ausschalten von RUN-STOP/RESTORE

Wie Sie vielleicht wissen, stehen in der Zeropage und der erweiterten Zeropage des Commodore 64 einige Vektoren. Das sind Zeiger, die normalerweise auf bestimmte Rom-Abschnitte weisen, die wir jedoch, da sie im Ram stehen, auf einfachste Weise manipulieren können. So auch den Zeiger auf die Stop-Routine.

Dieser kann mit

```
POKE 808,225
```

außer Kraft gesetzt werden, da der Vektor nun direkt hinter die Routine zeigt.

Ähnliches ist auch mit dem

```
POKE 792,193
```

möglich, nur das hier nun die RESTORE-Taste blockiert. Auch hier zeigt dann wieder der NMI-Vektor (ein solcher Non-Maskable-Interrupt wird durch die Restore-Taste nämlich ausgelöst) hinter die Routine, und kann so nicht mehr abgearbeitet werden.

8.3 Der BASIC-Editor als Textverarbeitung

Durch POKE 22,35 wird der Stringstack so manipuliert, daß die Ausgabe der Zeilennummern und das anschließende READY.

beim Listen gänzlich verschwindet. Sie haben nun die Möglichkeit Ihre Texte wie im folgenden Beispiel einzugeben:

```
10 Sehr geehrter Herr Mueller.
20 Bei unserem letzen Treffen haben Sie
30 leider Ihren Hut bei uns liegen-
40 gelassen....
 .
 .
```

Bei Eingabe des obenstehenden POKE und einem nachfolgenden LIST erscheint nun exakt derselbe Text - nur ohne Zeilennummer.

8.4 Ein funktionierendes RE-NEW

Durch Eingabe der folgenden Zeilen ist ein RENEW auch von Basic aus möglich:

```
POKE 2050,0     : Zeiger auf 1. BASIC-Zeile setzen
SYS 42291       : BASIC-Zeilen neu binden
POKE 46,PEEK(35)-(PEEK(781) 253) : LOW-Byte und HIGH-Byte
POKE 45,PEEK(781)+2 AND 25       : der Speicherobergrenze
                                 : neu setzen
```

8.5 "Formular to Complex" unterdrücken

Wer kennt das nicht? Auf einmal anwortet der C64 nur noch mit einem ständigen "FORMULAR TOO COMPLEX"-Error, und ist durch (fast) nichts mehr zu bewegen dieses zu unterlassen. Doch hier hilft ein

```
POKE 24,0
```

um den Commodore 64 seinen normalen Dienst wieder antreten zu lassen. Dann wird der Zeiger auf den letzten String, der ja wohl das Unheil angerichtet hat, zurückgebogen.

8.6 Verkürzen der Floppyzugriffszeit

Durch folgende Befehlsfolge können Sie die Zeitdauer, nach der ein Interrupt in der Floppy ausgelöst wird, "hochsetzen". Dadurch bewegt sich dann der Schrittmotor der Floppy wesentlich schneller. Sie erhalten so eine wesentlich kürzere Zugriffszeit.

```
open 1,8,15,"m-w"+chr$(7)+chr$(28)+chr$(1)+chr$(15)
```

8.7 Finden von Datazeilen

Und wieder eine Situation, die ständig vorkommt. Es erscheint ein SYNTAX ERROR in einer Daten-Einlese-Routine, die die Aufgabe hat, z.B. 1400 Datas einzulesen. Statt nun die gesamte Data-Wüste zu durchsuchen, können Sie die Data-Zeile, in der Ihr Fehler auftrat, durch

```
?PEEK(63)+256*PEEK(64)
```

auslesen. Diese beiden Speicherstellen enthalten nämlich die gerade gelesene Data-Zeile.

8.8 Mehr Struktur in BASIC-Programme

Durch das Einfügen von Grafikzeichen am Anfang einer Zeile können Sie Ihre Programme viel besser strukturieren. Der Basic-

Interpreter ignoriert diese Grafikzeichen beim Listen und Ablaufen eines Programms.

8.9 Input ohne Fragezeichen

Der Commodore 64 bietet Ihnen die Möglichkeit, die Tastatur, die auch als Pheripheriegerät betrachtet wird, als File zu öffnen. Bei einem solchen Zugriff wird der Input als eine Eingabe gesehen, die nicht vom Bildschirm kommt, und folglich gibt der Interpreter auch kein Fragezeichen aus. Ein kurzes Beispielprogramm:

```
100 OPEN 1,0
110 INPUT#1,a$
120 CLOSE 1
```

8.10 Einfache BASIC-Hardcopy

Genauso wie Sie die Tastatur als File öffnen können, kann dies ebenfalls mit dem Bildschirm geschehen. Mit dem folgenden kleinen Programm können Sie auf einfachste Weise eine Hardcopy vom Bildschirm anfertigen.

```
100 OPEN 1,3
110 OPEN 2,4
120 PRINT CHR$(19);
130 FOR Y=0 TO 24
140   FOR X=0 TO 39
140     GET#1,a$
150     PRINT#2,a$;
160   NEXT X
170   PRINT#2
180 NEXT Y
190 CLOSE 1:CLOSE 2
```

8.11 Beenden eines Programms ohne "READY."

Oft ist es von Nöten, die "READY"-Meldung nach Ablauf eines Programms zu unterdrücken. Schreiben Sie nun an das Ende eines Programms den Befehl

```
SYS 42144
```

so wird die READY-Meldung beim Durchführen der Warmstart-Routine "übergangen".

Weiterhin haben Sie die Möglichkeit, Ihre READY-Meldung ganz zu verändern. Voraussetzung dafür ist, daß der Basic-Interpreter im RAM steht. Mit folgender Zeile z.B. können Sie die READY-Meldung dann modifizieren.:

```
100 a$="Hallo"+chr$(13)
110 for x=1 to 6
120       poke 41847+x,asc(mid$(a$,1,1))
130 next
```

Beachten Sie hierbei bitte, daß die Zeichenkette a$ nicht länger als 6 Zeichen sein darf.

8.12 Prinzip eines Autostarts von Disketten

Wenn Sie Ihr Programm automatisch starten lassen wollen und ein wenig Begabung in Assemblerprogrammierung besitzen, so können Sie das folgende Prinzip in die Praxis umsetzen. Schreiben Sie (das geht nur mit einem Assembler) ein Programm, das bei $101 beginnt und spätestes bei $150 endet. Danach müssen Sie den Bereich von $151 bis $1ff mit $01-Bytes auffüllen. Hierdurch wird beim Laden Ihres Laders, der Stack überschrieben. Nachdem nun das Programm geladen wurde, wird

die Laderoutine mit RTS verlassen, es werden also 2 Bytes vom Stack geholt. Da aber der Stack überschrieben wurde, wird der neue Program-Counter nun auf die Adresse $0101 zeigen, wo Sie nun weitere Operationen durchführen lassen können.

8.13 Programmierung des Modulstarts

Es ist schon schade wenn man ein sehr langes Programm geschrieben hat und dieses dann von Programm-Knackern einfach kopiert wird. Diesen Leuten können Sie die Arbeit ziemlich erschweren, indem Sie den Resettaster Ihres Computers blockieren. Das kann mit Hilfe des folgenden Programms geschehen:

```
100 a$="CBM80": rem unbedingt groß schreiben
100 poke 8*4096,0:poke 8*4096,0
110 for x=8*4096+3 to 8*4096+8
120 poke x,asc(mid$(a$,x-8*4096+3,1))
130 next x
```

Der Resetzeiger zeigt nun auf die Speicherstelle 0, die einen illegalen Assemblerbefehl enthält und unter normalen Umständen - den Prozessor sofort abstürzen läßt.

8.14 Wenn der Computer sich weigert zu rechnen

Das ist Ihnen doch bestimmt auch oft passiert. Aufeinmal ist es nicht mehr möglich, dem Computer auch nur eine einzige Funktion zu entlocken. Das liegt daran, daß der JMP-Code, der als Vektor auf Funktionen dient, in der Zeropage liegt und durch wildes herumPOKEn leicht überschrieben werden kann. Durch

```
POKE 84,4*16+12
```

wird jedoch der Originalzustand wiederherstellt.

8.15 Verhindern der LIST-Funktion

Natürlich gewähren auch Sie nicht gerne Einblick in Ihre Programme. Doch oft weiß niemand dies zu verhindern. Durch

```
POKE 775,199
```

können Sie die List-Funktion "unschädlich machen". Hierbei wird dann nämlich der Vektor für List verbogen, so daß die List-Routine nicht mehr ausgeführt werden kann.

8.16 Vernünftiges Laden von Assemblerprogrammen

Auch der Fehler des Betriebssystems, das Laden von Assemblerprogrammen dem Laden von Basic-Programmen fast gleichzusetzen, kann behoben werden. Durch folgende Befehle können Sie Ihr Assemblerprogramm laden und später trotzdem noch mit Ihrem Basic-programm weiterarbeiten:

```
poke 250,peek(45)
poke 251,peek(46)
load "name",8,1
poke 45,peek(250)
poke 46,peek(251)
clr
```

8.17 Das Beschleunigen von Programmen

Wenn Sie sehr zeitkritische Dinge in Basic durchführen, so können Sie Ihr Programm etwas beschleunigen, indem Sie den

Bildschirm ausschalten (natürlich softwaremäßig). Dies kann durch folgenden POKE geschehen:

```
POKE 53248+17,11
```

Ein Anschalten hingegen, geschieht mit folgendem POKE:

```
POKE 53248+17,27
```

8.18 Steuerung der Datasette

Sie selbst können die Datasette steuern. Mit folgenden POKEs können Sie den Motor Ihrer Datasette aus- und einschalten. Dadurch wäre z. B. auch ein Directory mit Kassette möglich, da ja nun Ihr Programm die Steuerung der Datasette übernimmt.

```
POKE 0,47   : Motor ein
POKE 0,15   : Motor aus
```

8.19 Erzwingen von Fehlermeldungen

Wenn Sie gerne, aus welchen Gründen auch immer, in Ihrem Basic-programm eine beliebige Fehlermeldung des Interpreters ausgeben lassen möchten, so erreichen Sie dies durch:

```
poke 781,x
sys 42039
```

Hierdurch wird das X-Register nach dem Aufruf mit SYS mit dem Wert geladen, der in der Speicherstelle 781 steht. Dann

wird in die Routine gesprungen, die den Fehler schließlich ausgibt und das Basic-programm unterbricht.

8.20 Den Cursor frisieren

Sollte Ihnen die Geschwindigkeit, mit der sich der Cursor bewegt zu langsam sein, so können Sie durch

```
POKE 56325,X
```

die Geschwindigkeit mit dem ein Interrupt ausgelöst wird und so auch die Geschwindigket des Cursors beschleunigen. Sie sollten dabei X nicht größer als 10 wählen, weil sich der Computer sonst in einer Endlosschleife aufhängen kann.

8.21 Kopie des Video-Controllers

Sehr nützlich kann es zu wissen sein, daß es insgesamt 5 Kopien des Video-Controllers gibt. Diese liegen direkt hinter dem Video-Controller und zwar an folgenden Adressen:

```
$D100, $D200, $D300
```

Ebenso wie für den Video-Controller, exestieren auch Kopien vom SID. Diese liegen ebenfalls hinter demselben an folgenden Adressen:

```
$D500, $D600, $D700
```

8.22 Ein simpler Kopierschutz

Stellen Sie sich einmal vor, Sie haben eine Diskette, deren Inhaltsverzeichnis folgendes besagt:

```
0    "Name"        01 2a
33   "filename"        seq
          .
          .
     .    .
          .
```

Kämen Sie auf den Gedanken, daß es sich bei diesem File um ein Programm handeln könnte? - Doch, es funktioniert. Sie können Ihr Programm mit der folgenden Zeile als Sequenzielles oder auch User-File abspeichern:

```
SAVE "PRGNAME,S,W",8
```

Wenn Sie nun ein solches File einladen möchten, verfahren Sie wie folgt:

```
LOAD "PRGNAME,S,R",8
```

Möchten Sie ein User-File statt eines Sequentiellen Files, so brauchen sie lediglich das 'S' durch ein 'U' zu ersetzen.

8.23 Scroll-Down in Basic

Bestimmt brauchten Sie auch schon oft eine Scroll-Down Routine, wußten jedoch nicht, wie eine solche zu programmieren ist. Doch Sie brauchen selbst keine eigene Scroll-Down Routine zu schreiben, denn der Commodore 64 hat bereits eine derartige eingebaut. Sie können das erkennen, wenn sie eine

Programmzeile erstellen, die länger als 40 Zeichen ist. Dann nämlich scrollt der Computer den Bildschirm ab der aktuellen Cursorposition um eine Zeile nach unten. So ein Einfügen können Sie nun auch mit der folgenden Befehlsfolge realisieren:

```
print chr$(19)+chr$(17)+chr$(157)+chr$(148):poke 218,152
```

8.24 Routine zum Kopieren des Roms ins Ram

Der Basic-interpreter hält eine Routine bereit, mit der man Blöcke beliebiger Länge kopieren kann. Diese Routine können wir uns nun zu Nutze machen, und so eine Kurz-Routine programmieren, die das Rom ins Ram kopiert:

```
poke 88,0 : poke 89,0 : poke 90,0
poke 91,0 : poke 95,0 : poke 96,160
sys 41919
```

8.25 Ermitteln der aktuellen Fileparameter

Wenn Sie ein File auf Diskette oder Kassette oder auch auf dem Drucker eröffnen, so können Sie durch verschiedene Speicheradressen die einzelnen Parameter ermitteln. Das hat den Nutzen, daß Sie in Ihren eigenen Programmen feststellen können, ob der Benutzer Ihres Programms über ein Kassetten oder Diskettenlaufwerk verfügt. Hier nun die entsprechenden Speicherstellen:

```
184     - laufende Filenummer
185     - laufende Sekundär-Adresse
186     - laufende Geräteadresse
```

Diese Speicherstellen können Sie nun bei Bedarf mit der Peek-Funktion auslesen.

8.26 Das Anhängen von BASIC-Programmen

Um ein Basic-Programm an ein anderes anzuhängen, können Sie folgende Routine benutzen. Zunächst müssen Sie das Programm mit den niedrigeren Zeilennummern eingeben. Danach geben Sie folgende POKEs ein um den Basic-Start an das Ende des ersten Programms zu setzen,

```
POKE 43,PEEK(45)-2:POKE 44,PEEK(46)
```

und laden dann das zweite Programm nach. Schließlich müssen Sie noch den Basic-Start durch

```
POKE 43,1:POKE 44,8
```

auf den ursprünglichen Wert zurücksetzen.

8.27 GET# mit fünffacher Geschwindigkeit

Wenn Sie diesen Disk-Trick durchgelesen haben, werden Sie sicher nie wieder den GET#-Befehl zum Lesen von Files anwenden. Das Betriebssystem beinhaltet nämlich eine interne Routine , die das entsprechende Äquivalent zum CMD-Befehl darstellt. Das bedeutet, daß diese Routine nicht die Ausgabekanäle für Files festlegt, sondern Eingabekanäle definiert. Diese läßt sich nun an Stelle von GET#-Befehlen einsetzen.

Verfahren Sie dazu folgendermaßen:

POKEn Sie zunächst die Filenummer mit

```
POKE 781,fn
```

in die Adresse 781, und starten anschließend die Routine mit

```
SYS 65478
```

Ab sofort werden die Variablen mit der normalen GET-Anweisung aus dem entsprechenden File gelesen. Der Geschwindigkeitszuwachs kommt deswegen zustande, weil das Betriebssystem in unserem Falle das Äquivalent zu CMD nicht jedesmal aufheben muß, wie es beim normalen GET# der Fall gewesen wäre. Statt dessen müssen wir nun jeweils am Ende der Daten-Eingabe-Routine noch den Befehl

```
SYS 65484
```

eingeben, um die Eingabe wieder zu normalisieren.

8.28 Ready-Modifikation

Mit nur wenigen Operationen kann man das C-64 Betriebssystem so verändern, daß statt des öden und langweiligen

```
READY.
```

Befehls eine interessante beliebige Meldung erscheint. Wie wäre es beispielsweise, wenn Sie Ihr Computer nach jedem Befehl mit

```
Was soll ich jetzt tun, ehrwürdiger Meister?
```

oder einfach einem

```
ok!
HI!
```

oder auch

```
HALLO!
```

anspricht.

Sie können natürlich auch Ihren eigenen Namen einsetzen und sich persönlich anreden lassen.

Eine weitere sinnvolle READY-Modifikation ist auch, den READY-Befehl völlig abzuschalten und auch den manchmal störenden Zeilenvorschub zu eliminieren. Statt

```
READY.
_
```

erscheint dann nur noch der blinkende Cursor nach jedem Befehl. All diese Änderungen werden durch ein kurzes Programm bewerkstelligt.

Wie funktioniert es? Zunächst wird in der Zeile 30 und 35 das Betriebssystem in den RAM kopiert. Anschließend müssen Sie

Ihren neuen READY-Text eingeben, beispielsweise "Hi!!" oder eine beliebige Mitteilung.

Wenn Sie hier nur RETURN betätigen ohne einen Text festzulegen, wird jegliche READY-Meldung unterdrückt, später erscheint dann nur noch der Cursor.

Anschließend können Sie noch bestimmen, ob nach der Meldung ein Zeilenvorschub durchgeführt werden soll oder ob nicht.

Die Zeile 80 bewirkt, daß der Computer den neuen READY-Text in den Adressen 828 ff sucht und nicht mehr an der bisher üblichen Speicherstelle.

Dann erfolgt in den Zeilen 90 bis 180 die Generierung der neuen, vorher eingegebenen Meldung. Viel Spaß mit Ihren neuen persönlichen READY-Statements.

Und hier kommt nun das Listing der Routine:

```
10 rem ready modifikation vers.6
20 poke 53272,23:poke 53280,0:poke 53281,0
25 print"■READY-Modifikation"spc(230)
27 print"1/2 Minute Geduld!"spc(100)
30 for j=40960 to 49151:poke j,peek(j):nex
t
35 poke 1,peek(1)and 254
40 for i=828 to 900:poke i,0:next i
50 input "■READY.-Text...> ";a$:printspc(8
0)
60 input "Anschliessend Zeilenvorschub (j/
n) ";b$
70 :
80 poke 42101,60:poke 42103,3
90 poke 828,13
100 poke 829,10
105 if a$="" then i=830:goto  150
110 for i=830 to 829+len(a$)
120 :    poke i,asc(mid$(a$,i-829,1))
```

```
130 next i
140 :
160 if b$="j" then poke i,13:i=i+1
170 poke i+1,10
180 poke i+2,0
```

8.29 INTformation

Vielleicht haben Sie bereits bemerkt, daß die INT-Funktion des C-64. So ergibt beispielsweise

```
INT(-3.6325)
```

das Ergebnis von -4, obwohl eine "echte" Integerfunktion hier das Ergebnis von -3 liefern soll.

Allerdings ist das Verhalten von INT bei negativen Werten auch eine Definitionssache.

Eine derartige echte Integerfunktion läßt sich jedoch folgendermaßen schnell generieren:

```
10 X=-3.6325
20 PRINT  INT( ABS(X)) * SGN (X)
```

Diese korrekte INTEGER-Funktion kann auch als Makro definiert werden.

Wo wir gerade bei INTEGER-Funktionen sind, möchte ich Ihnen hier noch eine Liste alle der Befehle geben, die automa-

tisch INT-Werte generieren, bevor weitere Verarbeitungen erfolgen. Es sind im einzelnen:

```
CHR$
DIM
GOSUB
GOTO
LEFT$
MID$
PEEK
POKE
RIGHT$
SPC
TAB
```

8.30 Unerlaubte Variablen

Bleiben wir noch kurz bei den Variablen. Es gibt im BASIC des C-64 "verbotene" Variablen, die einen Syntax-Error generieren.

Ein Beispiel dafür stellt das folgende Programm dar:

```
10 AUTOS=21
20 SCHIFF=13.7
30 PRINT AUTOS+SCHIFF
```

Die Variablen dieses Programms sind verboten, weil in AUTO das BASIC-Keywort "TO" und in SCHIFF das logische "IF" enthalten ist.

Es gibt jedoch eine Möglichkeit, derartige Variablennamen trotzdem in Programmen zu verwenden.

Der Trick besteht darin, einen Graphic-Character in den Namen zu integrieren. Unser Beispielprogramm lautet dann so:

```
10 AUT(Shift J)OS=21
20 SCHI(Shift J)FF=13.7
30 PRINT AUT(Shift J)OS + SCHI(Shift J)FF
```

Das Programm läuft fehlerfrei. Die Graphikzeichen der Variablen sind beim Auflisten des Programms auch nicht mehr zu entdecken.

8.31 End-Reset

Durch den Befehl

```
POKE 768,143
```

wird bewirkt, daß ein Programm beim Erreichen eines END-Befehls oder dem Programmende ein Kaltstart ausgeführt wird, der alle Daten löscht.

8.32 Funktionstasten-Splitting

Mit einem Trick kann man die Anzahl der auf dem C-64 vorhandenden Funktionstasten von den normalen acht auf ganze zwölf Tasten erweitern.

Die normalen Funktionstasten werden im Direktmodus oder aber im SHIFT-Modus beispielsweise durch die folgende Anweisung abgefragt:

```
10 GET A$: IF A$="" THEN 10
20 :
30 IF A$="F3" THEN STOP
40 IF A$="F4" THEN PRINT "OK!"
50 GOTO 10
```

Die vier neuen Funktionstasten spricht man an, indem man die Commodore Taste zusammen mit den Funktionstasten betätigt.

Der Computer kennt drei Tasten, die mit anderen Tasten gleichzeitig gedrückt werden können.

Dies sind

```
SHIFT
COMMODORE
CONTROL
```

Welche dieser Tasten gedrückt wird, wird in der Speicherstelle 653 festgehalten. So bedeutet in der Anweisung

```
PRINT PEEK(653) AND 1
```

das Ergebnis 0, daß keine Taste gedrückt wurde,
das Ergebnis 1, daß SHIFT betätigt wurde,

```
PRINT PEEK(653) AND 2
```

das Ergebnis 2, daß die Control-Taste betätigt wurde,

```
PRINT PEEK(653) AND 4
```

das Ergebnis 4, daß die Commodore-Taste gedrückt wurde und

```
PRINT PEEK(653) AND 7
```

überprüft alle drei Tasten gleichzeitig.

Über die Adresse 197 kann der Code der derzeit gedrückten Taste bestimmt werden. Kombiniert man nun die beiden Adressen-Abfragen, so kann man sich die zwölf Funktionstasten realisieren:

```
10 GET KEY$
20 IF PEEK(197)=5 AND PEEK(653)=0 THEN (F3 normal)
30 IF PEEK(197)=5 AND PEEK(653)=1 THEN (F3 SHIFT)
40 IF PEEK(197)=5 AND PEEK(653)=2 THEN (F3 COMMOD)
```

Der Code 5 steht dabei für die Funktionstaste Nummer drei. Die Funktionstasten haben die folgende Belegungen:

```
F1  ...  CODE 4
F3  ...  CODE 5
F5  ...  CODE 6
F7  ...  CODE 3
```

Wie in dem obigen Beispiel können entsprechend auch die F1, F3 und F7 Tasten dreifach belegt werden. Die jeweils dritte Aktion wird ausgelöst, indem das Commodore-Zeichen zusammen mit einer der vier Funktionstasten betätigt wird.

Die zwölf Funktionstasten lassen sich paradoxerweise jedoch nicht auf insgesamt sechzehn erweitern, indem man das Controll-Zeichen entsprechend dem Commodore-Zeichen abfragt. Doch sollten die so erhaltenen vier neuen Funktionstasten, also insgesamt zwölf, in Ihren Programmen vielfältige Anwendungen finden.

8.33 Time-Delay-Loops

Viele BASIC-Programme enthalten Zeitverzögerungsschleifen wie

```
10 PRINT "HALLO WIE..."
20 :  FOR A=1 TO 200
30 :  NEXT A
40 PRINT "...GEHTS?"
```

Diese weit verbreitete Methode, um ein Programm für einige Zeit zu verzögern, läßt sich jedoch noch eleganter realisieren. Hierzu verwendet man die JIFFI-CLOCK.

Ein Beispiel sieht so aus:

```
10 PRINT "HALLO WIE..."
20 :  A=TI
30 :  IF A<(TI+60*5) THEN 30
40 PRINT "...GEHTS?"
```

Der große Vorteil der unteren Verzögerungsschleife ist, daß die Verzögerungsdauer genau festgelegt werden kann.

Während bei FOR-NEXT Verzögerungsschleifen die Dauer der Pause mehr oder weniger ungewiß oder auf jeden Fall äußerst vage ist, haben Sie mittels der TIME-Verzögerung die Möglichkeit, die Pausenlänge genau auf Bruchteile von Sekunden festzulegen.

Die Zeile 30 bewirkt eine Pausenlänge von 5 Sekunden (60 * 1/60stel Sekunde * 5). Das gilt natürlich für die normale JIFFI-CLOCK, die nicht wie beschrieben modifiziert wurde, also alle 60stel Sekunden den Wert von TI erhöht.

Anhang

DIE TOKEN-REFERENZLISTE

Alle Token mit einem Wert kleiner als 128 lassen sich der ASCII-Liste entnehmen (Eine derartige Liste finden Sie beispielsweise im C-64 Handbuch auf S.135ff).

Beispielsweise entspricht das Zeichen "A" dem ASCII und Token Wert 65. Kompliziert wird es nur bei den Token-Wertengrößer als 127. Diese beinhalten die BASIC-Schlüsselwörter.

Als Nachschlagehilfe kommt daher für diese Fälle die Referenzliste:

CHR$()-CODE	ENTSPRECHENDER BASIC-TOKEN
128	END
129	FOR
130	NEXT
131	DATA
132	INPUT#
133	INPUT
134	DIM
135	READ
136	LET
137	GOTO
138	RUN
139	IF
140	RESTORE
141	GOSUB
142	RETURN
143	REM
144	STOP
145	ON
146	WAIT
147	LOAD
148	SAVE
149	VERIFY
150	DEF

151 ..... POKE
152 ..... PRINT#
153 ..... PRINT
154 ..... CONT
155 ..... LIST
156 ..... CLR
157 ..... CMD
158 ..... SYS
159 ..... OPEN
160 ..... CLOSE
161 ..... GET
162 ..... NEW
163 ..... TAB
164 ..... TO
165 ..... FN
166 ..... SPC(
167 ..... THEN
168 ..... NOT
169 ..... STEP
170 ..... +
171 ..... -
172 ..... *
173 ..... /
174 ..... ^
175 ..... AND
176 ..... OR
177 ..... >
178 ..... =
179 ..... <
180 ..... SGN
181 ..... INT
182 ..... ABS
183 ..... USR
184 ..... FRE
185 ..... POS
186 ..... SQR
187 ..... RND
188 ..... LOG
189 ..... EXP
190 ..... COS

191 ..... SIN
192 ..... TAN
193 ..... ATN
194 ..... PEEK
195 ..... LEN
196 ..... STR$
197 ..... VAL
198 ..... ASC
199 ..... CHR$
200 ..... LEFT$
201 ..... RIGHT$
202 ..... MID$

ASCII- und Bildschirmcodetabellen

Chr$ & Poke Kodes

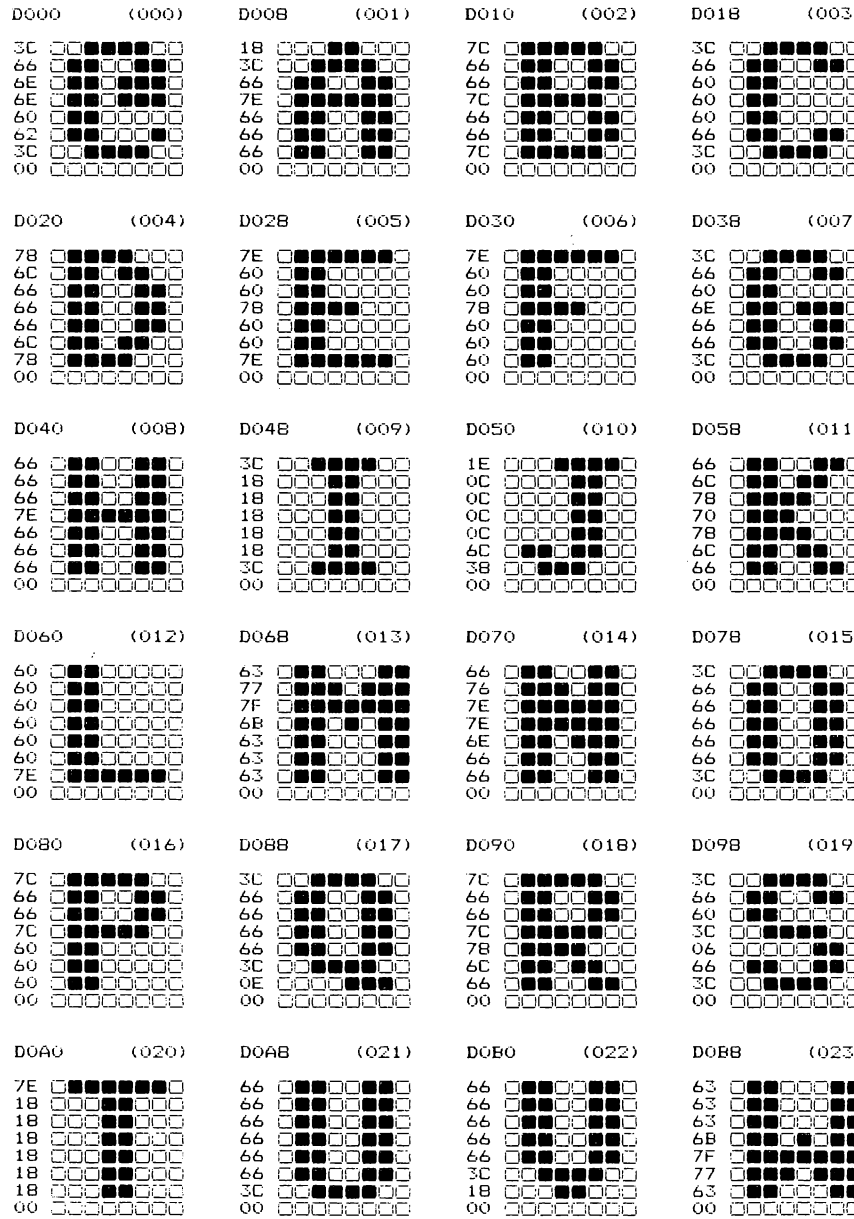
Dez	Hex	POKE- Zeichen Gross	Klein	CHR$- Zeichen Gross	CHR$- Zeichen Klein
000	$00	@	@		Keine Funktion
001	$01	A	a		Keine Funktion
002	$02	B	b		Keine Funktion
003	$03	C	c		Keine Funktion
004	$04	D	d		Keine Funktion
005	$05	E	e		Weiss
006	$06	F	f		Keine Funktion
007	$07	G	g		Keine Funktion
008	$08	H	h		Shift Commodore verboten
009	$09	I	i		Shift Commodore erlauben
010	$0a	J	j		Keine Funktion
011	$0b	K	k		Keine Funktion
012	$0c	L	l		Keine Funktion
013	$0d	M	m		Return
014	$0e	N	n		Kleinbuchstaben
015	$0f	O	o		Keine Funktion
016	$10	P	p		Keine Funktion
017	$11	Q	q		Cursor abwaerts
018	$12	R	r		Revers anschalten
019	$13	S	s		Cursor Home
020	$14	T	t		Delete
021	$15	U	u		Keine Funktion
022	$16	V	v		Keine Funktion
023	$17	W	w		Keine Funktion
024	$18	X	x		Keine Funktion
025	$19	Y	y		Keine Funktion
026	$1a	Z	z		Keine Funktion
027	$1b	[	[		Escape
028	$1c	£	£		Rot
029	$1d	]	]		Cursor rechts
030	$1e	↑	↑		Gruen
031	$1f	←	←		Dunkelblau
032	$20				Space
033	$21	!	!	!	!
034	$22	"	"	"	"
035	$23	#	#	#	#
036	$24	$	$	$	$
037	$25	%	%	%	%
038	$26	&	&	&	&
039	$27	'	'	'	'
040	$28	(	(	(	(
041	$29	)	)	)	)
042	$2a	*	*	*	*
043	$2b	+	+	+	+
044	$2c	,	,	,	,
045	$2d	-	-	-	-
046	$2e	.	.	.	.
047	$2f	/	/	/	/
048	$30	0	0	0	0
049	$31	1	1	1	1
050	$32	2	2	2	2
051	$33	3	3	3	3
052	$34	4	4	4	4
053	$35	5	5	5	5
054	$36	6	6	6	6
055	$37	7	7	7	7
056	$38	8	8	8	8
057	$39	9	9	9	9
058	$3a	:	:	:	:
059	$3b	;	;	;	;

060	$3c	<	<	<	<
061	$3d	=	=	=	=
062	$3e	>	>	>	>
063	$3f	?	?	?	?
064	$40	@	@	@	@
065	$41	A	A	A	A
066	$42	B	B	B	B
067	$43	C	C	C	C
068	$44	D	D	D	D
069	$45	E	E	E	E
070	$46	F	F	F	F
071	$47	G	G	G	G
072	$48	H	H	H	H
073	$49	I	I	I	I
074	$4a	J	J	J	J
075	$4b	K	K	K	K
076	$4c	L	L	L	L
077	$4d	M	M	M	M
078	$4e	N	N	N	N
079	$4f	O	O	O	O
080	$50	P	P	P	P
081	$51	Q	Q	Q	Q
082	$52	R	R	R	R
083	$53	S	S	S	S
084	$54	T	T	T	T
085	$55	U	U	U	U
086	$56	V	V	V	V
087	$57	W	W	W	W
088	$58	X	X	X	X
089	$59	Y	Y	Y	Y
090	$5a	Z	Z	Z	Z
091	$5b	[	[	[	[
092	$5c	£	£	£	£
093	$5d	]	]	]	]
094	$5e	↑	↑	↑	↑
095	$5f	←	←	←	←
096	$60				
097	$61				
098	$62				
099	$63				
100	$64				
101	$65				
102	$66				
103	$67				
104	$68				
105	$69				
106	$6a				
107	$6b				
108	$6c				
109	$6d				
110	$6e				
111	$6f				
112	$70				
113	$71				
114	$72				
115	$73				
116	$74				
117	$75				
118	$76				
119	$77				
120	$78				
121	$79				
122	$7a				
123	$7b				
124	$7c				
125	$7d				
126	$7e				
127	$7f				
128	$80			Keine Funktion	
129	$81			Keine Funktion	
130	$82			Keine Funktion	
131	$83			Keine Funktion	

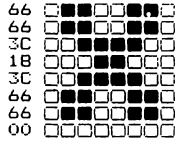
132	$84						Keine Funktion
133	$85						Weiss
134	$86						Keine Funktion
135	$87						Keine Funktion
136	$88						Shift Commodore verboten
137	$89						Shift Commodore erlauben
138	$8a						Keine Funktion
139	$8b						Keine Funktion
140	$8c						Keine Funktion
141	$8d						Return
142	$8e						Kleinbuchstaben
143	$8f						Keine Funktion
144	$90						Keine Funktion
145	$91						Cursor abwaerts
146	$92						Revers anschalten
147	$93						Cursor Home
148	$94						Delete
149	$95						Keine Funktion
150	$96						Keine Funktion
151	$97						Keine Funktion
152	$98						Keine Funktion
153	$99						Keine Funktion
154	$9a						Keine Funktion
155	$9b						Escape
156	$9c						Rot
157	$9d						Cursor rechts
158	$9e						Gruen
159	$9f						Dunkelblau
160	$a0						Space
161	$a1				!		!
162	$a2				"		"
163	$a3				#		#
164	$a4				$		$
165	$a5				%		%
166	$a6				&		&
167	$a7				'		'
168	$a8				(		(
169	$a9				)		)
170	$aa				*		*
171	$ab				+		+
172	$ac				,		,
173	$ad				-		-
174	$ae				.		.
175	$af				/		/
176	$b0				0		0
177	$b1				1		1
178	$b2				2		2
179	$b3				3		3
180	$b4				4		4
181	$b5				5		5
182	$b6				6		6
183	$b7				7		7
184	$b8				8		8
185	$b9				9		9
186	$ba				:		:
187	$bb				;		;
188	$bc				<		<
189	$bd				=		=
190	$be				>		>
191	$bf				?		?
192	$c0				@		@
193	$c1				A		a
194	$c2				B		b
195	$c3				C		c
196	$c4				D		d
197	$c5				E		e
198	$c6				F		f
199	$c7				G		g
200	$c8				H		h
201	$c9				I		i
202	$ca				J		j
203	$cb				K		k

204	$cc
205	$cd
206	$ce
207	$cf
208	$d0
209	$d1
210	$d2
211	$d3
212	$d4
213	$d5
214	$d6
215	$d7
216	$d8
217	$d9
218	$da
219	$db
220	$dc
221	$dd
222	$de
223	$df
224	$e0
225	$e1
226	$e2
227	$e3
228	$e4
229	$e5
230	$e6
231	$e7
232	$e8
233	$e9
234	$ea
235	$eb
236	$ec
237	$ed
238	$ee
239	$ef
240	$f0
241	$f1
242	$f2
243	$f3
244	$f4
245	$f5
246	$f6
247	$f7
248	$f8
249	$f9
250	$fa
251	$fb
252	$fc
253	$fd
254	$fe
255	$ff

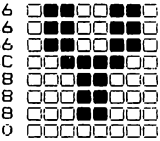
Kompletter Zeichensatz des C64



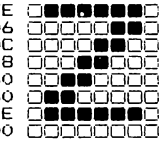
D0C0 (024)



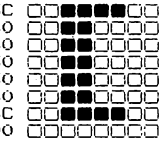
D0C8 (025)



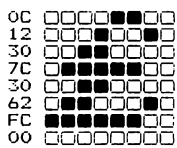
D0D0 (026)



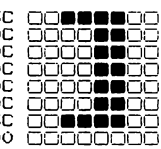
D0D8 (027)



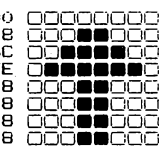
D0E0 (028)



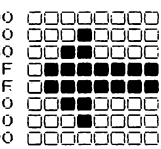
D0E8 (029)



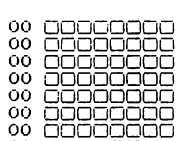
D0F0 (030)



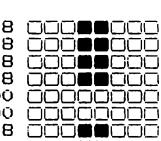
D0F8 (031)



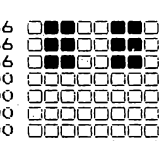
D100 (032)



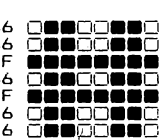
D108 (033)



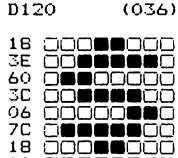
D110 (034)



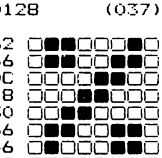
D118 (035)



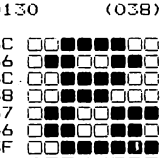
D120 (036)



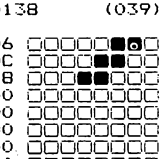
D128 (037)



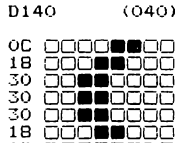
D130 (038)



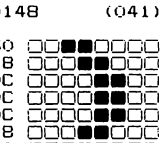
D138 (039)



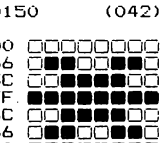
D140 (040)



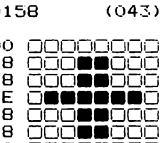
D148 (041)



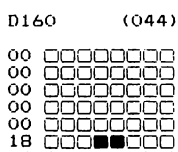
D150 (042)



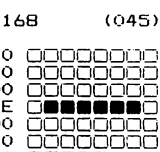
D158 (043)



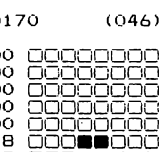
D160 (044)



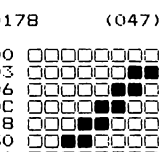
D168 (045)

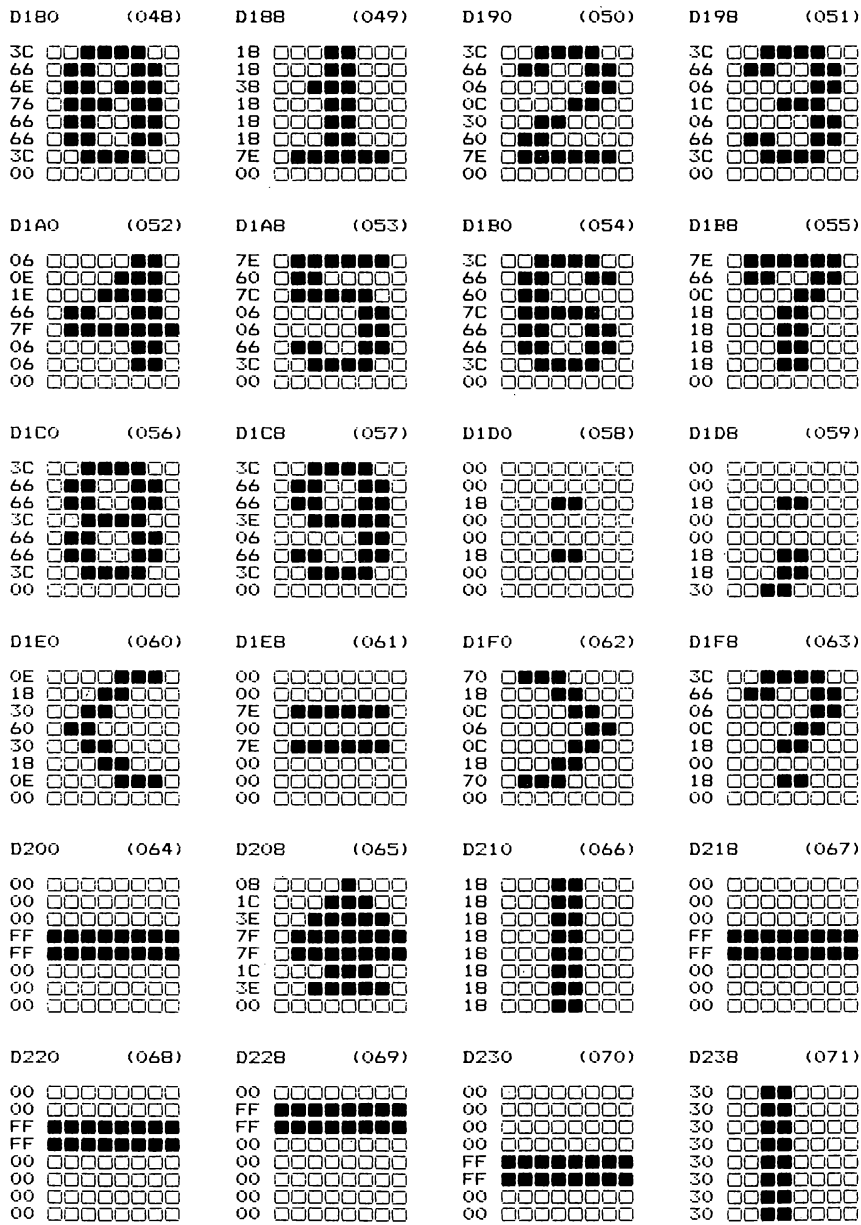


D170 (046)

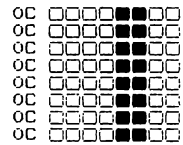


D178 (047)

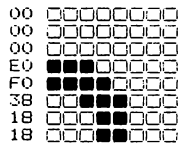



D180 (048)
3C
66
6E
76
66
66
3C
00
D188 (049)
18
18
38
18
18
18
7E
00
D190 (050)
3C
66
06
0C
30
60
7E
00
D198 (051)
3C
66
06
1C
06
66
3C
00
D1A0 (052)
06
0E
1E
66
7F
06
06
00
D1A8 (053)
7E
60
7C
06
06
66
3C
00
D1B0 (054)
3C
66
60
7C
66
66
3C
00
D1B8 (055)
7E
66
0C
18
18
18
18
00
D1C0 (056)
3C
66
66
3C
66
66
3C
00
D1C8 (057)
3C
66
66
3E
06
66
3C
00
D1D0 (058)
00
00
18
00
00
18
00
00
D1D8 (059)
00
00
18
00
00
18
18
30
D1E0 (060)
0E
18
30
60
30
18
0E
00
D1E8 (061)
00
00
7E
00
7E
00
00
00
D1F0 (062)
70
18
0C
06
0C
18
70
00
D1F8 (063)
3C
66
06
0C
18
00
18
00
D200 (064)
00
00
00
FF
FF
00
00
00
D208 (065)
08
1C
3E
7F
7F
1C
3E
00
D210 (066)
18
18
18
18
18
18
18
18
D218 (067)
00
00
00
FF
FF
00
00
00
D220 (068)
00
00
FF
FF
00
00
00
00
D228 (069)
00
FF
FF
00
00
00
00
00
D230 (070)
00
00
00
00
FF
FF
00
00
D238 (071)
30
30
30
30
30
30
30
30

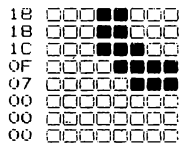
D240 (072)



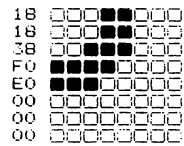
D248 (073)



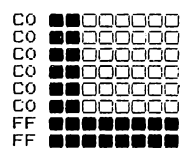
D250 (074)



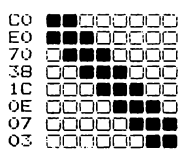
D258 (075)



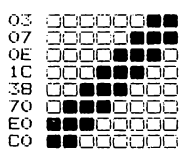
D260 (076)



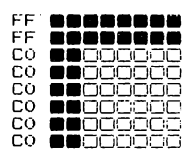
D268 (077)



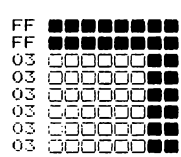
D270 (078)



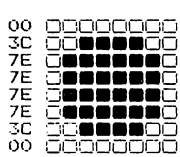
D278 (079)



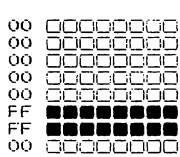
D280 (080)



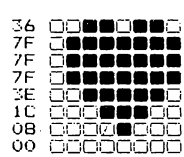
D288 (081)



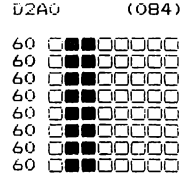
D290 (082)



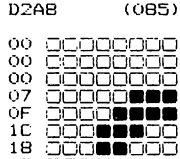
D298 (083)



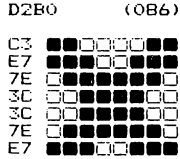
D2A0 (084)



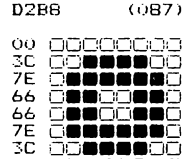
D2A8 (085)



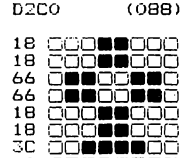
D2B0 (086)



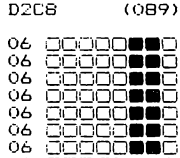
D2B8 (087)



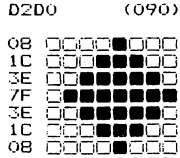
D2C0 (088)



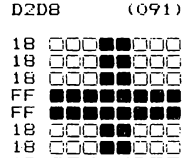
D2C8 (089)



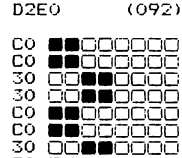
D2D0 (090)



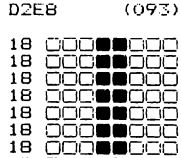
D2D8 (091)



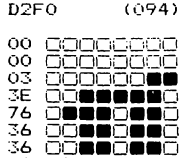
D2E0 (092)



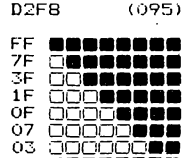
D2E8 (093)



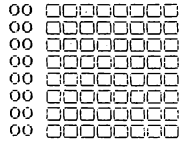
D2F0 (094)



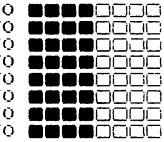
D2F8 (095)



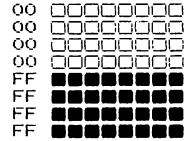
D300 (096)



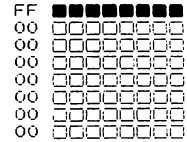
D308 (097)



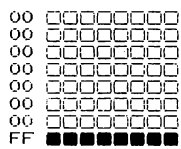
D310 (098)



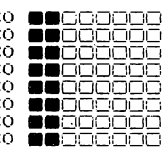
D318 (099)



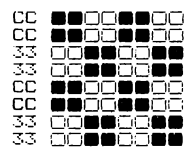
D320 (100)



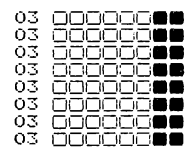
D328 (101)



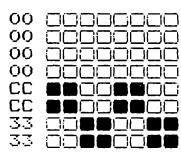
D330 (102)



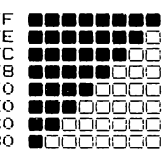
D338 (103)



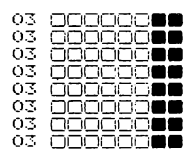
D340 (104)



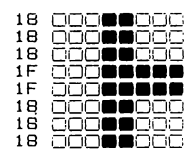
D348 (105)



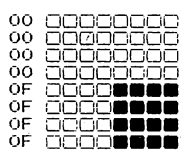
D350 (106)



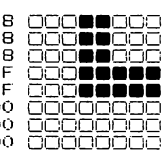
D358 (107)



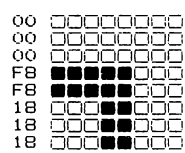
D360 (108)



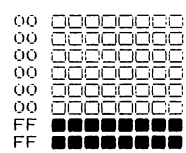
D368 (109)



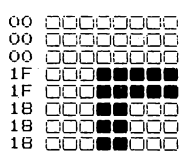
D370 (110)



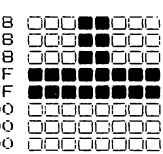
D378 (111)



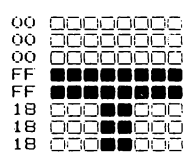
D380 (112)



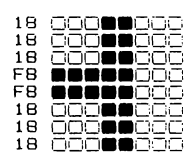
D388 (113)



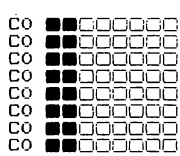
D390 (114)



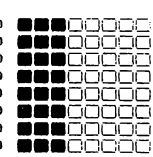
D398 (115)



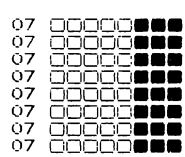
D3A0 (116)



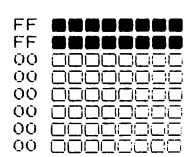
D3A8 (117)

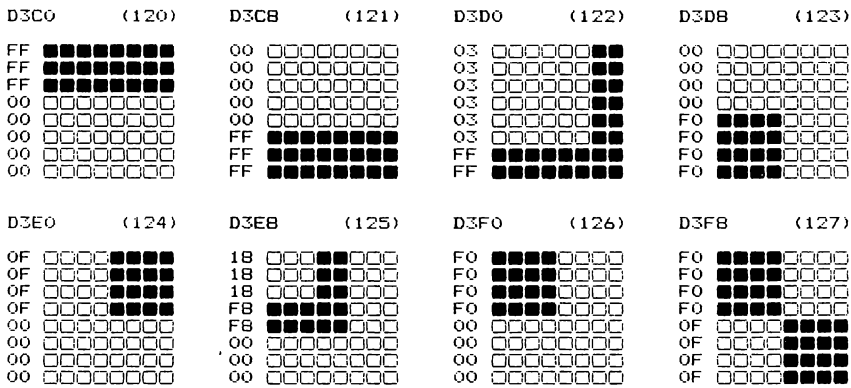


D3B0 (118)



D3B8 (119)




D3C0 (120)
FF
FF
FF
00
00
00
00
00
D3C8 (121)
00
00
00
00
00
FF
FF
FF
D3D0 (122)
03
03
03
03
03
FF
FF
FF
D3D8 (123)
00
00
00
00
F0
F0
F0
F0
D3E0 (124)
0F
0F
0F
0F
00
00
00
00
D3E8 (125)
18
18
18
F8
F8
00
00
00
D3F0 (126)
F0
F0
F0
F0
00
00
00
00
D3F8 (127)
F0
F0
F0
F0
0F
0F
0F
0F

Umrechnungstabelle: dezimal - hexadezimal - binär

dezimal	hex	binär	dezimal	hex	binär
0	00	00000000	26	1A	00011010
1	01	00000001	27	1B	00011011
2	02	00000010	28	1C	00011100
3	03	00000011	29	1D	00011101
4	04	00000100	30	1E	00011110
5	05	00000101	31	1F	00011111
6	06	00000110	32	20	00100000
7	07	00000111	33	21	00100001
8	08	00001000	34	22	00100010
9	09	00001001	35	23	00100011
10	0A	00001010	36	24	00100100
11	0B	00001011	37	25	00100101
12	0C	00001100	38	26	00100110
13	0D	00001101	39	27	00100111
14	0E	00001110	40	28	00101000
15	0F	00001111	41	29	00101001
16	10	00010000	42	2A	00101010
17	11	00010001	43	2B	00101011
18	12	00010010	44	2C	00101100
19	13	00010011	45	2D	00101101
20	14	00010100	46	2E	00101110
21	15	00010101	47	2F	00101111
22	16	00010110	48	30	00110000
23	17	00010111	49	31	00110001
24	18	00011000	50	32	00110010
25	19	00011001	51	33	00110011

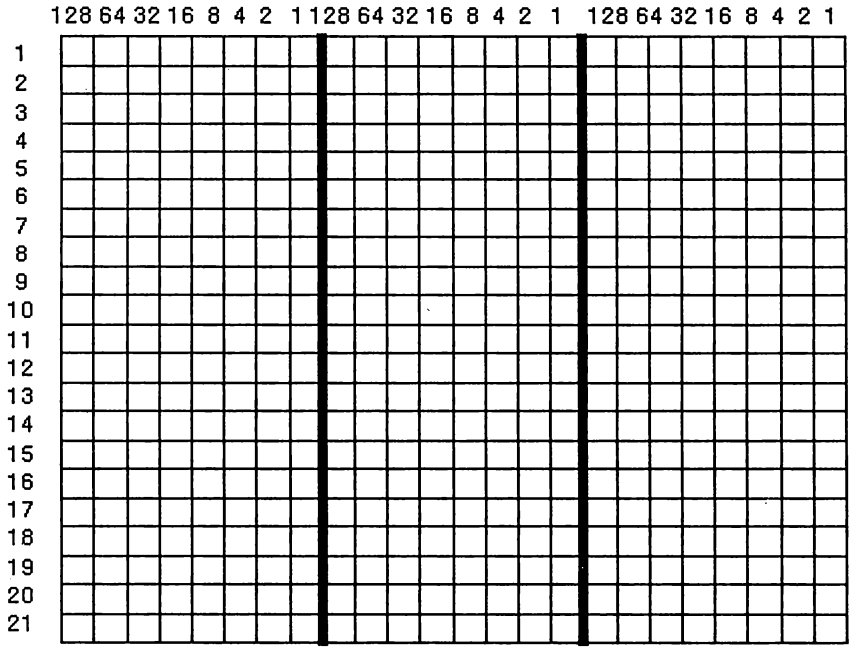
dezimal	hex	binär	dezimal	hex	binär
52	34	00110100	78	4E	01001110
53	35	00110101	79	4F	01001111
54	36	00110110	80	50	01010000
55	37	00110111	81	51	01010001
56	38	00111000	82	52	01010010
57	39	00111001	83	53	01010011
58	3A	00111010	84	54	01010100
59	3B	00111011	85	55	01010101
60	3C	00111100	86	56	01010110
61	3D	00111101	87	57	01010111
62	3E	00111110	88	58	01011000
63	3F	00111111	89	59	01011001
64	40	01000000	90	5A	01011010
65	41	01000001	91	5B	01011011
66	42	01000010	92	5C	01011100
67	43	01000011	93	5D	01011101
68	44	01000100	94	5E	01011110
69	45	01000101	95	5F	01011111
70	46	01000110	96	60	01100000
71	47	01000111	97	61	01100001
72	48	01001000	98	62	01100010
73	49	01001001	99	63	01100011
74	4A	01001010	100	64	01100100
75	4B	01001011	101	65	01100101
76	4C	01001100	102	66	01100110
77	4D	01001101	103	67	01100111

dezimal	hex	binär	dezimal	hex	binär
104	68	01101000	130	82	10000010
105	69	01101001	131	83	10000011
106	6A	01101010	132	84	10000100
107	6B	01101011	133	85	10000101
108	6C	01101100	134	86	10000110
109	6D	01101101	135	87	10000111
110	6E	01101110	136	88	10001000
111	6F	01101111	137	89	10001001
112	70	01110000	138	8A	10001010
113	71	01110001	139	8B	10001011
114	72	01110010	140	8C	10001100
115	73	01110011	141	8D	10001101
116	74	01110100	142	8E	10001110
117	75	01110101	143	8F	10001111
118	76	01110110	144	90	10010000
119	77	01110111	145	91	10010001
120	78	01111000	146	92	10010010
121	79	01111001	147	93	10010011
122	7A	01111010	148	94	10010100
123	7B	01111011	149	95	10010101
124	7C	01111100	150	96	10010110
125	7D	01111101	151	97	10010111
126	7E	01111110	152	98	10011000
127	7F	01111111	153	99	10011001
128	80	10000000	154	9A	10011010
129	81	10000001	155	9B	10011011

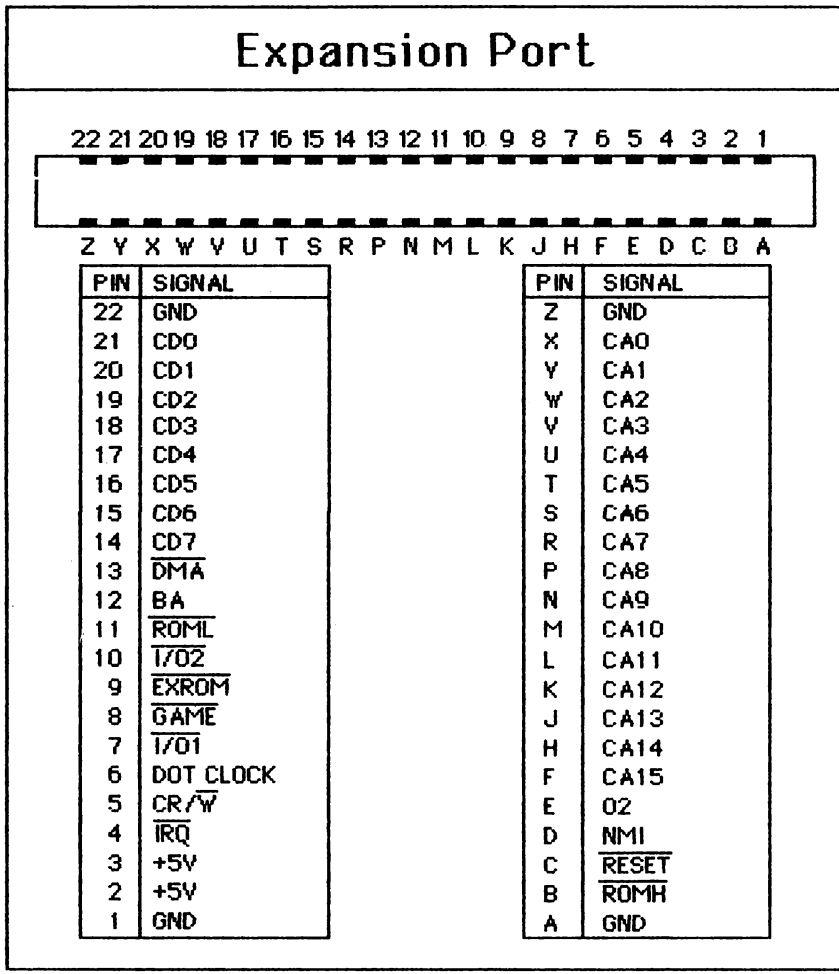
dezimal	hex	binär	dezimal	hex	binär
156	9C	10011100	182	B6	10110110
157	9D	10011101	183	B7	10110111
158	9E	10011110	184	B8	10111000
159	9F	10011111	185	B9	10111001
160	A0	10100000	186	BA	10111010
161	A1	10100001	187	BB	10111011
162	A2	10100010	188	BC	10111100
163	A3	10100011	189	BD	10111101
164	A4	10100100	190	BE	10111110
165	A5	10100101	191	BF	10111111
166	A6	10100110	192	C0	11000000
167	A7	10100111	193	C1	11000001
168	A8	10101000	194	C2	11000010
169	A9	10101001	195	C3	11000011
170	AA	10101010	196	C4	11000100
171	AB	10101011	197	C5	11000101
172	AC	10101100	198	C6	11000110
173	AD	10101101	199	C7	11000111
174	AE	10101110	200	C8	11001000
175	AF	10101111	201	C9	11001001
176	B0	10110000	202	CA	11001010
177	B1	10110001	203	CB	11001011
178	B2	10110010	204	CC	11001100
179	B3	10110011	205	CD	11001101
180	B4	10110100	206	CE	11001110
181	B5	10110101	207	CF	11001111

dezimal	hex	binär	dezimal	hex	binär
208	D0	11010000	234	EA	11101010
209	D1	11010001	235	EB	11101011
210	D2	11010010	236	EC	11101100
211	D3	11010011	237	ED	11101101
212	D4	11010100	238	EE	11101110
213	D5	11010101	239	EF	11101111
214	D6	11010110	240	F0	11110000
215	D7	11010111	241	F1	11110001
216	D8	11011000	242	F2	11110010
217	D9	11011001	243	F3	11110011
218	DA	11011010	244	F4	11110100
219	DB	11011011	245	F5	11110101
220	DC	11011100	246	F6	11110110
221	DD	11011101	247	F7	11110111
222	DE	11011110	248	F8	11111000
223	DF	11011111	249	F9	11111001
224	E0	11100000	250	FA	11111010
225	E1	11100001	251	FB	11111011
226	E2	11100010	252	FC	11111100
227	E3	11100011	253	FD	11111101
228	E4	11100100	254	FE	11111110
229	E5	11100101	255	FF	11111111
230	E6	11100110			
231	E7	11100111			
232	E8	11101000			
233	E9	11101001			

Sprite-Entwurfsblatt



Expansion Port



PIN	SIGNAL
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	$\overline{\text{DMA}}$
12	BA
11	$\overline{\text{ROML}}$
10	$\overline{\text{I/O2}}$
9	$\overline{\text{EXROM}}$
8	$\overline{\text{GAME}}$
7	$\overline{\text{I/O1}}$
6	DOT CLOCK
5	CR/$\overline{\text{W}}$
4	$\overline{\text{IRQ}}$
3	+5V
2	+5V
1	GND

PIN	SIGNAL
Z	GND
X	CA0
Y	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	02
D	NMI
C	$\overline{\text{RESET}}$
B	$\overline{\text{ROMH}}$
A	GND

User Port

User Port

PIN	SIGNAL	BEMERKUNG
1	GND	
2	+5V	MAX. 100 mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	

PIN	SIGNAL
A	GND
B	$\overline{\text{FLAG 2}}$
C	PB 0
D	PB 1
E	PB 2
F	PB 3
H	PB 4
J	PB 5
K	PB 6
L	PB 7
M	PA 2
N	GND

Chip: VIC 6567-Videocontroller

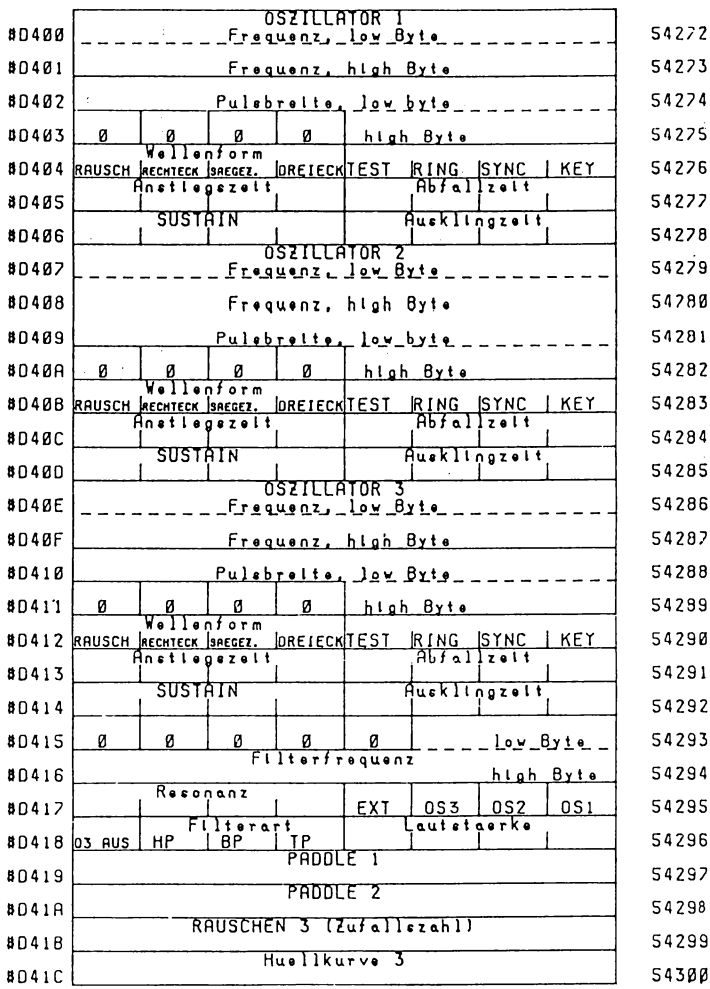
CHIP : VIC (6567) Video-Controller

+0	$D000	Sprite 0 X-Koordinate (0-255)								53248
+1	$D001	Y-Koordinate								53249
+2	$D002	Sprite 1 X-Koordinate (0-255)								53250
+3	$D003	Y-Koordinate								53251
+4	$D004	Sprite 2 X-Koordinate (0-255)								53252
+5	$D005	Y-Koordinate								53253
+6	$D006	Sprite 3 X-Koordinate (0-255)								53254
+7	$D007	Y-Koordinate								53255
+8	$D008	Sprite 4 X-Koordinate (0-255)								53256
+9	$D009	Y-Koordinate								53257
+10	$D00A	Sprite 5 X-Koordinate (0-255)								53258
+11	$D00B	Y-Koordinate								53259
+12	$D00C	Sprite 6 X-Koordinate (0-255)								53260
+13	$D00D	Y-Koordinate								53261
+14	$D00E	Sprite 7 X-Koordinate (0-255)								53262
+15	$D00F	Y-Koordinate								53263
+16	$D010	Sprite X-Koordinate +256								53264
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0	
+17	$D011	Raster RC8	Hintergr. Mehrfarb	Hi-Res Grafik	Bild aus	25/24 Zeilen	Position YSCL2	Position YSCL1	Position YSCL0	53265
+18	$D012	Raster RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	53266
+19	$D013	Light - Pen X-Koordinate								53267
+20	$D014	Light - Pen Y-Koordinate								53268
+21	$D015	Sprite an/aus								53269
		SE7	SE6	SE5	SE4	SE3	SE2	SE1	SE0	
+22	$D016	Nicht benutzt 0	0	0	Mehrfarb Modus	40/38 Spalten	Position XSCL2	Position XSCL1	Position XSCL0	53270
+23	$D017	Sprite vergroesserung Y-Koordinate								53271
		SEXY7	SEXY6	SEXY5	SEXY4	SEXY3	SEXY2	SEXY1	SEXY0	
+24	$D018	Bildschirmspeicher Adr. (o1024) CB13	CB12	CB11	CB10	Zeichengen. Adr. (o2048) VS13	VS12	VS11	Nicht benutzt	53272
+25	$D019	IRQ Flag	Nicht benutzt 0	0	0	Light-P. I. -Flag	Sp.-Sp. Koll.	Sp.-Hin. Koll.	Raster I. -Flag	53273
+26	$D01A	Nicht benutzt 0	0	0	0	Maske MLPI	Maske MISSC	Maske MISBC	Maske MRIRQ	53274
+27	$D01B	Sprite-Hintergrund Prioritaet								53275
		BSP7	BSP6	BSP5	BSP4	BSP3	BSP2	BSP1	BSP0	
+28	$D01C	Mehrfarbige Sprite								53276
		SMC7	SMC6	SMC5	SMC4	SMC3	SMC2	SMC1	SMC0	

+29	#D01D	Sprite vorgrossern X-Koordinate							53277
		SEXX7	SEXX6	SEXX5	SEXX4	SEXX3	SEXX2	SEXX1	SEXX0	
+30	#D01E	Sprite-Sprite Kollision							53278
		SSC7	SSC6	SSC5	SSC4	SSC3	SSC2	SSC1	SSC0	
+31	#D01F	Sprite-Hintergrund Kollision							53279
		SBC7	SBC6	SBC5	SBC4	SBC3	SBC2	SBC1	SBC0	
+32	#D020	Randfarbe (0-15)							53280
		EXTJ							
+33	#D021	Hintergrundfarbe Nr. 0 (0-15)							53281
		BKGD0							
+34	#D022	Hintergrundfarbe Nr. 1 (0-15)							53282
		BKGD1							
+35	#D023	Hintergrundfarbe Nr. 2 (0-15)							53283
		BKGD2							
+36	#D024	Hintergrundfarbe Nr. 3 (0-15)							53284
		BKGD3							
37	#D025	Gemeinsame Sprite-Farbe Nr. 0 im Mehrfarben-Modus (0-15)							53285
		SMC0							
+38	#D026	Gemeinsame Sprite-Farbe Nr. 1 im Mehrfarben-Modus (0-15)							53286
		SMC1							
+39	#D027	Farbe Sprite Nr. 0 (0-15)							53287
		S0COL							
+40	#D028	Farbe Sprite Nr. 1 (0-15)							53288
		S1COL							
+41	#D029	Farbe Sprite Nr. 2 (0-15)							53289
		S2COL							
+42	#D02A	Farbe Sprite Nr. 3 (0-15)							53290
		S3COL							
+43	#D02B	Farbe Sprite Nr. 4 (0-15)							53291
		S4COL							
+44	#D02C	Farbe Sprite Nr. 5 (0-15)							53292
		S5COL							
+45	#D02D	Farbe Sprite Nr. 6 (0-15)							53293
		S6COL							
+46	#D02E	Farbe Sprite Nr. 7 (0-15)							53294
		S7COL							

Chip: SID 6561-Soundsynthesizer

CHIP : SID (6581) Sound Synthesizer



Chip: CIA 6526

		PRA							
+0	$DC00	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	56320
		PRB							
+1	$DC01	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	56321
		DDRA							
+2	$DC02	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0	56322
		DDRB							
+3	$DC03	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0	56323
		TA LO							
+4	$DC04	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0	56324
		TA MI							
+5	$DC05	TAM7	TAM6	TAM5	TAM4	TAM3	TAM2	TAM1	TAM0	56325
		TB LO							
+6	$DC06	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0	56326
		TB MI							
+7	$DC07	TBM7	TBM6	TBM5	TBM4	TBM3	TBM2	TBM1	TBM0	56327
		TOD 10THS							
+8	$DC08	0	0	0	0	T0	T4	T2	T1	56328
		TOD SEC							
+9	$DC09	0	SH4	SH2	SH1	SL0	SL4	SL2	SL1	56329
		TOD MIN							
+10	$DC0A	0	MH4	MH2	MH1	ML0	ML4	ML2	ML1	56330
		TOD HR							
+11	$DC0B	PM	0	0	HH	HL0	HL4	HL2	HL1	56331
		SDR							
+12	$DC0C	S7	S6	S5	S4	S3	S2	S1	S0	56332
		ICR							
+13	$DC0D	IRQ	0	0	FLG	SP	ALRM	TB	TA	56333
		CRA							
+14	$DC0E	SP MI	SP MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START	56334
		CRB							
+15	$DC0F	ALARM	IN MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START	56335

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 1

120:   C000                        .OPT P1,00
120:   C000                        .TIT "SPRITE-EDITOR V1.0"
120:   C000                        .SYM 4
130:                         ;
140:   C000                        *=   49152
150:                         ;
160:   C000 20 65 C3 ANF     JSR  INIT
160:   C003 A9 20            LDA  #$20
160:   C005 8D 00 C0         STA  ANF
160:   C008 A9 62            LDA  #<BASI
160:   C00A 8D 01 C0         STA  ANF+1
160:   C00D A9 C7            LDA  #>BASI
160:   C00F 8D 02 C0         STA  ANF+2
170:   C012 4C FE C3         JMP  TASTER
180:                         ;
190:   0057           XK     =    $57
190:   0058           YK     =    $58
190:   0059           CURCOL =    $59
200:   0018           GRUNDX =    24
200:   0042           GRUNDY =    66
210:   FFE4           GET    =    $FFE4
220:   0478           WER1   =    1144
220:   A320           WER2   =    41760
230:                         ;
240:                         ;****************************************
250:                         ;*** "SET COLOR" ***********************
260:                         ;****************************************
270:                         ;
280:   C015 A9 78    SCOLOR LDA  #<55416
280:   C017 A0 DB            LDY  #>55416
290:   C019 85 FA            STA  $FA
290:   C01B 84 FB            STY  $FB
290:   C01D A5 F7            LDA  247
300:                         ;
310:   C01F A2 15    SKIP1  LDX  #21
320:   C021 A0 18    LAB1   LDY  #24
330:   C023 91 FA    LAB2   STA  ($FA),Y
340:   C025 88               DEY
350:   C026 D0 FB            BNE  LAB2
350:   C028 48               PHA
350:   C029 20 31 C0         JSR  PLUSC40
350:   C02C 68               PLA
350:   C02D CA               DEX
350:   C02E D0 F1            BNE  LAB1
350:   C030 60               RTS
360:                         ;
370:   C031 A5 FA    PLUSC40 LDA  $FA
370:   C033 18               CLC
370:   C034 69 28            ADC  #40
370:   C036 90 02            BCC  OK
380:   C038 E6 FB            INC  $FB
390:   C03A 85 FA    OK     STA  $FA
400:   C03C 60               RTS
410:                         ;
420:                         ;****************************************
430:                         ;*** "SPRITE LOESCHEN" ******************
440:                         ;****************************************
450:                         ;
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 2

460:   C03D A9 78    LOESCH   LDA  #<1144
460:   C03F A0 04             LDY  #>1144
470:   C041 85 FA             STA  $FA
470:   C043 84 FB             STY  $FB
470:   C045 A9 20             LDA  #32
470:   C047 4C 1F C0          JMP  SKIP1
480:                  ;
490:                  ;**************************************
500:                  ;*** "SPRITE INVERTIEREN" *************
510:                  ;**************************************
520:                  ;
530:   C04A A9 78    INVERT   LDA  #<1144
530:   C04C A0 04             LDY  #>1144
540:   C04E 85 FA             STA  $FA
540:   C050 84 FB             STY  $FB
550:                  ;
560:   C052 A2 15             LDX  #21
570:   C054 A0 18    LAB3     LDY  #24
580:   C056 B1 FA    LAB4     LDA  ($FA),Y
580:   C058 49 80             EOR  #$80
590:   C05A 91 FA             STA  ($FA),Y
600:   C05C 88                DEY
610:   C05D D0 F7             BNE  LAB4
620:                  ;
630:   C05F 20 31 C0          JSR  PLUSC40
640:   C062 CA                DEX
650:   C063 D0 EF             BNE  LAB3
660:   C065 4C 15 C0          JMP  SCOLOR
670:                  ;
680:                  ;**************************************
690:                  ;*** "SCROLL UP" **********************
700:                  ;**************************************
710:                  ;
720:   C068 20 B6 C0 SCROLO   JSR  GETWER1
720:   C06B A0 19             LDY  #25
720:   C06D 20 78 C0          JSR  SLAB1
730:   C070 20 A5 C0          JSR  GETWER2
730:   C073 A0 19             LDY  #25
730:   C075 4C 78 C0          JMP  SLAB1
740:                  ;
750:   C078 B1 FA    SLAB1    LDA  ($FA),Y
750:   C07A 99 C7 C0          STA  BUF-1,Y
760:   C07D 88                DEY
770:   C07E D0 F8             BNE  SLAB1
770:   C080 20 E2 C0          JSR  PLUSB40
770:   C083 20 31 C0          JSR  PLUSC40
780:                  ;
790:   C086 A2 14             LDX  #20
800:   C088 A0 18    SLAB2    LDY  #24
810:   C08A B1 FA    SLAB3    LDA  ($FA),Y
820:   C08C 91 FC             STA  ($FC),Y
830:   C08E 88                DEY
840:   C08F D0 F9             BNE  SLAB3
850:                  ;
860:   C091 20 31 C0          JSR  PLUSC40
860:   C094 20 E2 C0          JSR  PLUSB40
870:   C097 CA                DEX
880:   C098 D0 EE             BNE  SLAB2
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 3

880:   C09A A0 18              LDY  #24
890:                      ;
900:   C09C B9 C7 C0 SLAB4     LDA  BUF-1,Y
900:   C09F 91 FC              STA  ($FC),Y
910:   C0A1 88                 DEY
910:   C0A2 D0 F8              BNE  SLAB4
910:   C0A4 60                 RTS
920:                      ;
930:   C0A5 A9 78     GETWER2  LDA  #<55416
930:   C0A7 A0 D8              LDY  #>55416
940:   C0A9 85 FA              STA  $FA
940:   C0AB 84 FB              STY  $FB
950:   C0AD A9 50              LDA  #<55416-40
950:   C0AF A0 D8              LDY  #>55416-40
960:   C0B1 85 FC              STA  $FC
960:   C0B3 84 FD              STY  $FD
960:   C0B5 60                 RTS
970:                      ;
980:   C0B6 A9 78     GETWER1  LDA  #<1144
980:   C0B8 A0 04              LDY  #>1144
990:   C0BA 85 FA              STA  $FA
990:   C0BC 84 FB              STY  $FB
1000:  C0BE A9 50              LDA  #<1144-40
1000:  C0C0 A0 04              LDY  #>1144-40
1010:  C0C2 85 FC              STA  $FC
1010:  C0C4 84 FD              STY  $FD
1010:  C0C6 60                 RTS
1020:                     ;
1030:  C0C7 00                 .BYT 0
1040:  C0C8               BUF  *=   *+26
1050:                     ;
1060:  C0E2 A5 FC         PLUSB40  LDA  $FC
1060:  C0E4 18                 CLC
1060:  C0E5 69 28              ADC  #40
1070:  C0E7 90 02              BCC  PBOK
1070:  C0E9 E6 FD              INC  $FD
1080:  C0EB 85 FC         PBOK     STA  $FC
1080:  C0ED 60                 RTS
1090:                     ;
1100:                     ;****************************************
1110:                     ;*** "SCROLL DOWN" ********************
1120:                     ;****************************************
1130:                     ;
1140:  C0EE A9 36         SCROLU   LDA  #54
1140:  C0F0 85 01              STA  1
1140:  C0F2 20 FA C0           JSR  SCUU
1140:  C0F5 A9 37              LDA  #55
1140:  C0F7 85 01              STA  1
1140:  C0F9 60                 RTS
1150:  C0FA 20 4A C1 SCUU      JSR  RETPIC
1160:  C0FD 20 6B C1           JSR  GETWER5
1160:  C100 20 31 C0           JSR  PLUSC40
1160:  C103 20 12 C1           JSR  SCU
1160:  C106 20 7C C1           JSR  GETWER6
1170:  C109 20 31 C0           JSR  PLUSC40
1170:  C10C 20 12 C1           JSR  SCU
1170:  C10F 4C 26 C1           JMP  OZEILE
1180:  C112 A2 14         SCU      LDX  #20
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 4

1180:  C114 A0 18    SULAB1   LDY  #24
1190:  C116 B1 FC    SULAB2   LDA  ($FC),Y
1190:  C118 91 FA             STA  ($FA),Y
1200:  C11A 88                DEY
1200:  C11B D0 F9             BNE  SULAB2
1210:                ;
1220:  C11D 20 31 C0          JSR  PLUSC40
1220:  C120 20 E2 C0          JSR  PLUSB40
1230:  C123 CA                DEX
1240:  C124 D0 EE             BNE  SULAB1
1250:                ;
1260:  C126 A0 18    OZEILE   LDY  #24
1270:  C128 B9 20 A3 OZLAB1   LDA  $A000+800,Y
1270:  C12B 99 78 04          STA  1144,Y
1280:  C12E B9 20 A8          LDA  $A500+800,Y
1280:  C131 99 78 D8          STA  55416,Y
1290:  C134 88                DEY
1290:  C135 D0 F1             BNE  OZLAB1
1300:  C137 60                RTS
1310:                ;
1320:  C138 A9 A0             LDA  #<1144+40
1320:  C13A A0 04             LDY  #>1144+40
1330:  C13C 85 FA             STA  $FA
1330:  C13E 84 FB             STY  $FB
1330:  C140 60                RTS
1340:                ;
1350:  C141 A9 A0             LDA  #<55416+40
1350:  C143 A0 D8             LDY  #>55416+40
1360:  C145 85 FA             STA  $FA
1360:  C147 84 FB             STY  $FB
1360:  C149 60                RTS
1370:                ;
1380:                ;*** PICTURE RETTEN ***
1390:                ;
1400:  C14A 20 6B C1 RETPIC   JSR  GETWER5
1400:  C14D 20 56 C1          JSR  RET1
1410:  C150 20 7C C1          JSR  GETWER6
1410:  C153 4C 56 C1          JMP  RET1
1420:                ;
1430:  C156 A2 15    RET1     LDX  #21
1440:  C158 A0 18    RETLAB1  LDY  #24
1450:  C15A B1 FA    RETLAB2  LDA  ($FA),Y
1450:  C15C 91 FC             STA  ($FC),Y
1460:  C15E 88                DEY
1460:  C15F D0 F9             BNE  RETLAB2
1470:  C161 20 31 C0          JSR  PLUSC40
1470:  C164 20 E2 C0          JSR  PLUSB40
1480:  C167 CA                DEX
1490:  C168 D0 EE             BNE  RETLAB1
1500:  C16A 60                RTS
1510:                ;
1520:  C16B A9 78    GETWER5  LDA  #<1144
1520:  C16D A0 04             LDY  #>1144
1530:  C16F 85 FA             STA  $FA
1530:  C171 84 FB             STY  $FB
1540:  C173 A9 00             LDA  #0
1540:  C175 A0 A0             LDY  #$A0
1550:  C177 85 FC             STA  $FC
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 5

1550:  C179 84 FD              STY  $FD
1550:  C17B 60                 RTS
1560:                    ;
1570:  C17C A9 78     GETWER6  LDA  #<55416
1570:  C17E A0 D8              LDY  #>55416
1580:  C180 85 FA              STA  $FA
1580:  C182 84 FB              STY  $FB
1590:  C184 A9 00              LDA  #$00
1590:  C186 A0 A5              LDY  #$A5
1600:  C188 85 FC              STA  $FC
1600:  C18A 84 FD              STY  $FD
1600:  C18C 60                 RTS
1610:                    ;
1620:                    ;****************************************
1630:                    ;*** "SCROLL LEFT" ********************
1640:                    ;****************************************
1650:                    ;
1660:  C18D 20 B6 C0  SCROLL   JSR  GETWER1
1660:  C190 20 99 C1           JSR  LSCROL
1670:  C193 20 A5 C0           JSR  GETWER2
1670:  C196 4C 99 C1           JMP  LSCROL
1680:                    ;
1690:  C199 A2 15     LSCROL   LDX  #21
1700:  C19B A0 01     SLLAB1   LDY  #1
1710:  C19D B1 FA              LDA  ($FA),Y
1710:  C19F 8D C8 C0           STA  BUF
1710:  C1A2 C8                 INY
1720:  C1A3 B1 FA     SLLAB2   LDA  ($FA),Y
1720:  C1A5 88                 DEY
1730:  C1A6 91 FA              STA  ($FA),Y
1730:  C1A8 C8                 INY
1730:  C1A9 C8                 INY
1740:  C1AA C0 19              CPY  #25
1740:  C1AC D0 F5              BNE  SLLAB2
1750:                    ;
1760:  C1AE AD C8 C0           LDA  BUF
1760:  C1B1 88                 DEY
1760:  C1B2 91 FA              STA  ($FA),Y
1770:  C1B4 20 31 C0           JSR  PLUSC40
1780:  C1B7 CA                 DEX
1790:  C1B8 D0 E1              BNE  SLLAB1
1800:  C1BA 60                 RTS
1810:                    ;
1820:                    ;****************************************
1830:                    ;*** "SCROLL RIGHT" *******************
1840:                    ;****************************************
1850:                    ;
1860:  C1BB 20 B6 C0  SCROLR   JSR  GETWER1
1860:  C1BE 20 C7 C1           JSR  RSCROL
1870:  C1C1 20 A5 C0           JSR  GETWER2
1870:  C1C4 4C C7 C1           JMP  RSCROL
1880:                    ;
1890:  C1C7 A2 15     RSCROL   LDX  #21
1900:  C1C9 A0 18     SRLAB1   LDY  #24
1910:  C1CB B1 FA              LDA  ($FA),Y
1910:  C1CD 8D C8 C0           STA  BUF
1910:  C1D0 88                 DEY
1920:  C1D1 B1 FA     SRLAB2   LDA  ($FA),Y
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 6

1920:  C1D3 C8                 INY
1930:  C1D4 91 FA              STA  ($FA),Y
1930:  C1D6 88                 DEY
1930:  C1D7 88                 DEY
1940:  C1D8 D0 F7              BNE  SRLAB2
1950:                  ;
1960:  C1DA AD C8 C0           LDA  BUF
1960:  C1DD C8                 INY
1960:  C1DE 91 FA              STA  ($FA),Y
1970:  C1E0 20 31 C0           JSR  PLUSC40
1980:  C1E3 CA                 DEX
1990:  C1E4 D0 E3              BNE  SRLAB1
2000:  C1E6 60                 RTS
2010:                  ;
2020:                  ;*** UEBERNEHMEN  ***
2030:                  ;
2040:  C1E7 00         SCOUNT  .BYT 0
2040:  C1E8 00         DREI    .BYT 0
2050:  C1E9 00         BITMUS  .BYT 0
2050:  C1EA 00         ZEILE   .BYT 0
2060:  0340            SPRITE  =    832
2070:                  ;
2080:  C1EB 80 40 20   MUSTER  .BYT 128,64,32,16,8,4,2,1
2090:                  ;
2100:  C1F3 20 15 C3   UEBERN  JSR  UMRECH
2100:  C1F6 20 B6 C0           JSR  GETWER1
2100:  C1F9 A9 00              LDA  #0
2100:  C1FB 8D E7 C1           STA  SCOUNT
2100:  C1FE 8D EA C1           STA  ZEILE
2100:  C201 8D E9 C1           STA  BITMUS
2100:  C204 18                 CLC
2110:  C205 A9 03      ULAB3   LDA  #3
2110:  C207 8D E8 C1           STA  DREI
2120:  C20A A2 00              LDX  #0
2120:  C20C A0 01              LDY  #1
2130:                  ;
2140:  C20E B1 FA      ULAB2   LDA  ($FA),Y
2150:  C210 C9 20              CMP  #" "
2150:  C212 F0 0A              BEQ  UOK1
2160:  C214 BD EB C1           LDA  MUSTER,X
2160:  C217 18                 CLC
2160:  C218 6D E9 C1           ADC  BITMUS
2170:  C21B 8D E9 C1           STA  BITMUS
2180:  C21E C8         UOK1    INY
2180:  C21F E8                 INX
2190:  C220 E0 08              CPX  #8
2190:  C222 D0 EA              BNE  ULAB2
2200:                  ;
2210:  C224 98                 TYA
2210:  C225 48                 PHA
2210:  C226 AC E7 C1           LDY  SCOUNT
2210:  C229 AD E9 C1           LDA  BITMUS
2220:  C22C 91 F8              STA  ($F8),Y
2220:  C22E 99 40 03           STA  SPRITE,Y
2220:  C231 EE E7 C1           INC  SCOUNT
2220:  C234 68                 PLA
2220:  C235 A8                 TAY
2220:  C236 A9 00              LDA  #0
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 7

2220:  C238 8D E9 C1          STA  BITMUS
2220:  C23B A2 00             LDX  #0
2230:  C23D CE E8 C1          DEC  DREI
2240:  C240 D0 CC             BNE  ULAB2
2250:                    ;
2260:  C242 20 31 C0          JSR  PLUSC40
2260:  C245 EE EA C1          INC  ZEILE
2260:  C248 AD EA C1          LDA  ZEILE
2270:  C24B C9 15             CMP  #21
2270:  C24D 90 B6             BCC  ULAB3
2280:  C24F A5 F7             LDA  247
2280:  C251 AC E7 C1          LDY  SCOUNT
2280:  C254 91 F8             STA  ($F8),Y
2280:  C256 60                RTS
2290:                    ;
2300:                    ;*** GET SPRITE ***
2310:                    ;
2320:  C257 20 B6 C0          JSR  GETWER1
2330:  C25A A9 00             LDA  #0
2330:  C25C A0 A0             LDY  #$A0
2340:  C25E 85 FC             STA  $FC
2340:  C260 84 FD             STY  $FD
2350:  C262 20 73 C2          JSR  GETS
2360:                    ;
2370:  C265 20 A5 C0          JSR  GETWER2
2380:  C268 A9 00             LDA  #0
2380:  C26A A0 A5             LDY  #$A5
2390:  C26C 85 FC             STA  $FC
2390:  C26E 84 FD             STY  $FD
2400:  C270 4C 73 C2          JMP  GETS
2410:                    ;
2420:  C273 A2 15        GETS LDX  #21
2430:  C275 A0 18        GLAB1 LDY  #24
2440:  C277 B1 FC        GLAB2 LDA  ($FC),Y
2440:  C279 91 FA             STA  ($FA),Y
2450:  C27B 88                DEY
2450:  C27C D0 F9             BNE  GLAB2
2460:                    ;
2470:  C27E 20 31 C0          JSR  PLUSC40
2470:  C281 20 E2 C0          JSR  PLUSB40
2480:  C284 CA                DEX
2490:  C285 D0 EE             BNE  GLAB1
2500:  C287 60                RTS
2510:                    ;
2520:                    ;*** CLEAR UR-SPRITE ***
2530:                    ;
2540:  C288 A9 00        CLRROM LDA  #0
2540:  C28A A0 A0             LDY  #$A0
2550:  C28C 85 FA             STA  $FA
2550:  C28E 84 FB             STY  $FB
2560:  C290 A9 00             LDA  #0
2560:  C292 A0 A5             LDY  #$A5
2570:  C294 85 FC             STA  $FC
2570:  C296 84 FD             STY  $FD
2580:                    ;
2590:  C298 A2 04             LDX  #4
2590:  C29A
2600:  C29A A0 00             LDY  #0
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 8

2610:  C29C A9 20     CSRLAB   LDA  #" "
2610:  C29E 91 FA              STA  ($FA),Y
2620:  C2A0 A9 00              LDA  #0
2620:  C2A2 91 FC              STA  ($FC),Y
2630:  C2A4 C8                 INY
2640:  C2A5 D0 F5              BNE  CSRLAB
2650:  C2A7 E6 FB              INC  $FB
2650:  C2A9 E6 FD              INC  $FD
2660:  C2AB CA                 DEX
2670:  C2AC D0 EE              BNE  CSRLAB
2680:  C2AE 60        FERTIG   RTS
2690:                 ;
2700:                 ;*** SPRITE KLEIN > GROSS ***
2710:                 ;
2720:  C2AF A5 02     KFGF     LDA  2        ;BLOCK NUMMER
2730:  C2B1 C9 81              CMP  #129
2730:  C2B3 B0 F9              BCS  FERTIG
2730:  C2B5 78                 SEI
2730:  C2B6 A9 35              LDA  #53
2730:  C2B8 85 01              STA  1
2740:                 ;
2750:  C2BA 20 B6 C0  UMWNDL    JSR  GETWER1
2750:  C2BD 20 15 C3            JSR  UMRECH
2750:  C2C0 A9 00               LDA  #0
2760:  C2C2 8D E7 C1            STA  SCOUNT
2760:  C2C5 8D EA C1            STA  ZEILE
2770:  C2C8 A9 03     WLAB3     LDA  #3
2770:  C2CA 8D E8 C1            STA  DREI
2770:  C2CD A0 01               LDY  #1
2780:  C2CF 98        WLAB1     TYA
2780:  C2D0 48                  PHA
2780:  C2D1 AC E7 C1            LDY  SCOUNT
2790:                 ;
2800:  C2D4 B1 F8               LDA  ($F8),Y
2800:  C2D6 99 40 03            STA  832,Y
2800:  C2D9 8D E9 C1            STA  BITMUS
2800:  C2DC 68                  PLA
2800:  C2DD A8                  TAY
2810:                 ;
2820:  C2DE A2 00               LDX  #0
2830:  C2E0 BD EB C1 WLAB2      LDA  MUSTER,X
2830:  C2E3 2D E9 C1            AND  BITMUS
2840:  C2E6 F0 03               BEQ  WOK2
2840:  C2E8 A9 A0               LDA  #160
2840:  C2EA 2C                  .BYT $2C
2850:  C2EB A9 20     WOK2      LDA  #32
2860:  C2ED 91 FA               STA  ($FA),Y
2860:  C2EF C8                  INY
2860:  C2F0 E8                  INX
2870:  C2F1 E0 08               CPX  #8
2870:  C2F3 D0 EB               BNE  WLAB2
2880:                 ;
2890:  C2F5 EE E7 C1            INC  SCOUNT
2890:  C2F8 CE E8 C1            DEC  DREI
2900:  C2FB D0 D2               BNE  WLAB1
2910:                 ;
2920:  C2FD 20 31 C0            JSR  PLUSC40
2920:  C300 20 E2 C0            JSR  PLUSB40
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 9

2920:  C303 EE EA C1          INC  ZEILE
2920:  C306 AD EA C1          LDA  ZEILE
2930:  C309 C9 15             CMP  #21
2930:  C30B 90 BB             BCC  WLAB3
2940:  C30D A9 37             LDA  #55
2940:  C30F 85 01             STA  1
2940:  C311 58                CLI
2940:  C312 4C 15 C0          JMP  SCOLOR
2950:                ;
2960:                ;*** UMRECHNUNG ***
2970:                ;
2980:  C315 C6 02    UMRECH   DEC  2
2980:  C317 8A                TXA
2980:  C318 48                PHA
2980:  C319 A5 02             LDA  2
2980:  C31B A2 00             LDX  #0
2980:  C31D 86 F8             STX  $F8
2980:  C31F 86 F9             STX  $F9
2990:  C321 85 F8             STA  $F8
3000:  C323 A2 06             LDX  #6
3000:  C325 18                CLC
3010:  C326 06 F8    UMRLAB   ASL  $F8
3010:  C328 26 F9             ROL  $F9
3020:  C32A CA                DEX
3030:  C32B D0 F9             BNE  UMRLAB
3040:  C32D A5 F8             LDA  $F8
3040:  C32F 18                CLC
3040:  C330 69 00             ADC  #<$E000
3040:  C332 90 02             BCC  UMROK
3050:  C334 E6 F9             INC  $F9
3050:  C336 85 F8    UMROK    STA  $F8
3050:  C338 A5 F9             LDA  $F9
3050:  C33A 18                CLC
3050:  C33B 69 E0             ADC  #>$E000
3050:  C33D 85 F9             STA  $F9
3050:  C33F 68                PLA
3050:  C340 AA                TAX
3050:  C341 E6 02             INC  2
3050:  C343 60                RTS
3060:                ;
3070:                ;***************************************
3080:                ;*** + UMSCHALTEN *********************
3090:                ;***************************************
3100:                ;
3110:  C344 E6 02    PLUM     INC  2
3110:  C346 A5 02             LDA  2
3110:  C348 18                CLC
3110:  C349 C9 80             CMP  #128
3110:  C34B 90 04             BCC  PLOK
3120:  C34D A9 01             LDA  #1
3120:  C34F 85 02             STA  2
3130:  C351 20 AF C2 PLOK     JSR  KFGF
3130:  C354 4C BD C3          JMP  NUMOUT
3140:                ;
3150:                ;***************************************
3160:                ;*** - UMSCHALTEN *********************
3170:                ;***************************************
3180:                ;
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 10

3190:  C357 C6 02    MIUM     DEC  2
3190:  C359 D0 04             BNE  MIOK
3200:  C35B A9 7F             LDA  #127
3200:  C35D 85 02             STA  2
3210:  C35F 20 AF C2 MIOK     JSR  KFGF
3210:  C362 4C BD C3          JMP  NUMOUT
3220:                ;
3230:                ;*** ROUTINEN INITIALISIEREN ***
3240:                ;
3250:  C365 A9 07    INIT     LDA  #7
3250:  C367 8D 15 D0          STA  53248+21
3260:  C36A A9 05             LDA  #5
3260:  C36C 8D 27 D0          STA  53248+39
3260:  C36F 8D 28 D0          STA  53248+40
3270:  C372 A9 F8             LDA  #248
3270:  C374 8D 00 D0          STA  53248
3270:  C377 8D 02 D0          STA  53250
3280:  C37A A9 83             LDA  #131
3280:  C37C 8D 01 D0          STA  53249
3290:  C37F A9 B1             LDA  #177
3290:  C381 8D 03 D0          STA  53251
3300:  C384 A9 0D             LDA  #13
3300:  C386 8D F8 07          STA  2040
3300:  C389 8D F9 07          STA  2041
3300:  C38C A9 0B             LDA  #11
3300:  C38E 8D FA 07          STA  2042
3300:  C391 A9 01             LDA  #1
3300:  C393 8D 29 D0          STA  53248+41
3310:  C396 A9 02             LDA  #2
3310:  C398 8D 17 D0          STA  53248+23
3310:  C39B 8D 1D D0          STA  53248+29
3320:  C39E 20 2E C4          JSR  INCURS
3320:  C3A1 A9 13             LDA  #19
3330:  C3A3 A9 01             LDA  #1
3330:  C3A5 85 02             STA  2
3330:  C3A7 A9 00             LDA  #0
3330:  C3A9 8D 20 D0          STA  $D020
3330:  C3AC 8D 21 D0          STA  $D021
3330:  C3AF 20 9C C4          JSR  CUROUT
3330:  C3B2 20 D0 C4          JSR  SCOLOUT
3340:  C3B5 20 DD C5          JSR  LLAB1
3340:  C3B8 A9 80             LDA  #128
3340:  C3BA 8D 8A 02          STA  650
3350:  C3BD A4 02    NUMOUT   LDY  2
3350:  C3BF 20 A2 B3          JSR  $B3A2
3350:  C3C2 20 DD BD          JSR  $BDDD
3360:  C3C5 A0 1A             LDY  #26
3360:  C3C7 A2 03             LDX  #3
3360:  C3C9 86 D6             STX  $D6
3360:  C3CB 84 D3             STY  $D3
3360:  C3CD 20 6C E5          JSR  $E56C
3370:  C3D0 A9 F6             LDA  #<T1
3370:  C3D2 A0 C3             LDY  #>T1
3370:  C3D4 20 1E AB          JSR  $AB1E
3380:  C3D7 A2 01             LDX  #1
3380:  C3D9 BD 00 01 NLAB1    LDA  $100,X
3380:  C3DC C9 00             CMP  #0
3380:  C3DE F0 0B             BEQ  ENDD
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 11

3390:  C3E0 20 D2 FF           JSR  $FFD2
3390:  C3E3 E8                 INX
3390:  C3E4 4C D9 C3           JMP  NLAB1
3400:  C3E7 A9 01              LDA  #1
3400:  C3E9 85 59              STA  CURCOL
3410:  C3EB A9 20     ENDD     LDA  #" "
3410:  C3ED 20 D2 FF NLAB2     JSR  $FFD2
3420:  C3F0 E8                 INX
3420:  C3F1 E0 05              CPX  #5
3420:  C3F3 D0 F8              BNE  NLAB2
3420:  C3F5 60                 RTS
3430:                 ;
3440:  C3F6 53 50 52 T1        .ASC "SPRT.#:"
3440:  C3FD 00                 .BYT 0
3450:                 ;
3460:                 ;*** TASTERTURABFRAGE ***
3470:                 ;
3480:  C3FE 20 E4 FF TASTER    JSR  GET
3480:  C401 F0 FB              BEQ  TASTER
3490:  C403 A2 00              LDX  #0
3500:  C405 DD AC C6 TASTLAB   CMP  TASTAB,X
3510:  C408 F0 08              BEQ  FOUND
3520:  C40A E8                 INX
3520:  C40B E0 1B              CPX  #27
3520:  C40D D0 F6              BNE  TASTLAB
3530:  C40F 4C FE C3           JMP  TASTER
3540:                 ;
3550:  C412 8A        FOUND    TXA
3550:  C413 0A                 ASL
3550:  C414 AA                 TAX
3560:  C415 BD C5 C6           LDA  WERTAB,X
3560:  C418 85 FA              STA  $FA
3570:  C41A BD C6 C6           LDA  WERTAB+1,X
3570:  C41D 85 FB              STA  $FB
3580:  C41F A9 C4              LDA  #>(RUECK-1)
3580:  C421 48                 PHA
3580:  C422 A9 27              LDA  #<(RUECK-1)
3580:  C424 48                 PHA
3590:  C425 6C FA 00           JMP  ($00FA)
3600:  C428 20 F3 C1 RUECK     JSR  UEBERN
3600:  C42B 4C FE C3           JMP  TASTER
3610:                 ;
3620:                 ;*** CURSOR INITIALISIEREN ***
3630:                 ;
3640:  C42E A2 00     INCURS   LDX  #0
3640:  C430 8A                 TXA
3640:  C431 9D C0 02 LOLAB     STA  704,X
3640:  C434 E8                 INX
3640:  C435 E0 40              CPX  #64
3640:  C437 D0 F8              BNE  LOLAB
3650:  C439 A9 FF              LDA  #255
3650:  C43B A2 12              LDX  #6*3
3660:  C43D 9D C0 02 CSLAB     STA  704,X
3660:  C440 E8                 INX
3660:  C441 E8                 INX
3660:  C442 E8                 INX
3670:  C443 E0 18              CPX  #8*3
3670:  C445 90 F6              BCC  CSLAB
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 12

3670:  C447 A9 01           LDA  #1
3670:  C449 85 57           STA  XK
3680:  C44B 85 58           STA  YK
3680:  C44D 85 59           STA  CURCOL
3680:  C44F A9 05           LDA  #5
3680:  C451 85 F7           STA  247
3680:  C453 20 59 C4        JSR  CUR
3680:  C456 4C 15 C0        JMP  SCOLOR
3690:                ;
3700:                ;*** CURSOR ABFRAGE ***
3710:                ;
3720:                ;--- BEREICH TESTEN
3730:                ;
3740:  C459 A5 57    CUR    LDA  XK
3740:  C45B C9 01           CMP  #1
3740:  C45D B0 06           BCS  ABOK1
3750:  C45F A9 18           LDA  #24
3750:  C461 85 57           STA  XK
3750:  C463 C6 58           DEC  YK
3760:  C465 C9 19    ABOK1  CMP  #25
3760:  C467 90 06           BCC  ABOK2
3770:  C469 A9 01           LDA  #1
3770:  C46B 85 57           STA  XK
3770:  C46D E6 58           INC  YK
3780:                ;
3790:  C46F A5 58    ABOK2  LDA  YK
3790:  C471 C9 01           CMP  #1
3790:  C473 B0 04           BCS  ABOK3
3800:  C475 A9 15           LDA  #21
3800:  C477 85 58           STA  YK
3810:  C479 C9 16    ABOK3  CMP  #22
3810:  C47B 90 04           BCC  ABOK4
3820:  C47D A9 01           LDA  #1
3820:  C47F 85 58           STA  YK
3830:                ;
3840:                ;--- CURSOR POSITIONIEREN
3850:                ;
3860:  C481 A5 57    ABOK4  LDA  XK
3860:  C483 0A              ASL
3860:  C484 0A              ASL
3860:  C485 0A              ASL          ; MAL 8
3870:  C486 18              CLC
3870:  C487 69 18           ADC  #GRUNDX
3870:  C489 48              PHA
3880:                ;
3890:  C48A A5 58           LDA  YK
3890:  C48C 0A              ASL
3890:  C48D 0A              ASL
3890:  C48E 0A              ASL          ; MAL 8
3900:  C48F 18              CLC
3900:  C490 69 42           ADC  #GRUNDY
3900:  C492 8D 05 D0        STA  53249+4
3910:  C495 68              PLA
3910:  C496 8D 04 D0        STA  53248+4
3920:                ;
3930:  C499 4C 6F C5        JMP  ZSOUT
3940:                ;
3950:                ;*** CURSOR FARBE AUSGEBEN ***
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 13

3960:                          ;
3970:  C49C A5 59     CUROUT   LDA  CURCOL
3970:  C49E C9 10              CMP  #16
3970:  C4A0 90 04              BCC  OOK1
3970:  C4A2 A9 00              LDA  #0
3970:  C4A4 85 59              STA  CURCOL
3970:  C4A6 A8         OOK1    TAY
3970:  C4A7 20 A2 B3           JSR  $B3A2
3980:  C4AA 20 DD BD           JSR  $BDDD
3990:  C4AD A0 1A              LDY  #26
3990:  C4AF A2 04              LDX  #4
3990:  C4B1 86 D6              STX  $D6
3990:  C4B3 84 D3              STY  $D3
4000:  C4B5 20 6C E5           JSR  $E56C
4010:  C4B8 A9 06              LDA  #<T2
4010:  C4BA A0 C5              LDY  #>T2
4010:  C4BC 20 1E AB           JSR  $AB1E
4020:  C4BF A2 01              LDX  #1
4030:  C4C1 BD 00 01 COLAB     LDA  $100,X
4030:  C4C4 F0 07              BEQ  COEND
4040:  C4C6 20 D2 FF           JSR  $FFD2
4040:  C4C9 E8                 INX
4040:  C4CA 4C C1 C4           JMP  COLAB
4050:                          ;
4060:  C4CD 4C EB C3 COEND     JMP  ENDD
4070:                          ;
4080:  C4D0 A5 F7     SCOLOUT  LDA  247
4080:  C4D2 C9 10              CMP  #16
4080:  C4D4 90 04              BCC  COOK1
4080:  C4D6 A9 00              LDA  #0
4080:  C4D8 85 F7              STA  247
4080:  C4DA A8         COOK1   TAY
4080:  C4DB 20 A2 B3           JSR  $B3A2
4090:  C4DE 20 DD BD           JSR  $BDDD
4100:  C4E1 A0 1A              LDY  #26
4100:  C4E3 A2 05              LDX  #5
4100:  C4E5 86 D6              STX  $D6
4100:  C4E7 84 D3              STY  $D3
4110:  C4E9 20 6C E5           JSR  $E56C
4120:  C4EC A9 0E              LDA  #<T3
4120:  C4EE A0 C5              LDY  #>T3
4120:  C4F0 20 1E AB           JSR  $AB1E
4130:  C4F3 A2 01              LDX  #1
4140:  C4F5 BD 00 01 COLAB2    LDA  $100,X
4140:  C4F8 C9 00              CMP  #0
4140:  C4FA F0 07              BEQ  CO2END
4150:  C4FC 20 D2 FF           JSR  $FFD2
4150:  C4FF E8                 INX
4150:  C500 4C F5 C4           JMP  COLAB2
4160:                          ;
4170:  C503 4C EB C3 CO2END    JMP  ENDD
4180:                          ;
4190:  C506 43 2D 43 T2        .ASC "C-COL :"
4190:  C50D 00                 .BYT 0
4200:  C50E 53 2D 43 T3        .ASC "S-COL :"
4200:  C515 00                 .BYT 0
4210:                          ;
4220:                          ;**************************************
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 14

4230:                  ;*** PUNKT LOESCHEN ******************
4240:                  ;**************************************
4250:                  ;
4260:  C516 A9 20      PER       LDA  #32
4270:  C518 2C                   .BYT $2C
4280:                  ;
4290:                  ;**************************************
4300:                  ;*** PUNKT SETZEN *********************
4310:                  ;**************************************
4320:                  ;
4330:  C519 A9 A0      PSET      LDA  #160
4330:  C51B 48                   PHA
4340:  C51C A5 57                LDA  XK
4340:  C51E 18                   CLC
4340:  C51F 48                   PHA
4350:  C520 A6 58                LDX  YK
4350:  C522 CA                   DEX
4350:  C523 8A                   TXA
4350:  C524 0A                   ASL
4350:  C525 AA                   TAX
4360:                  ;
4370:                  ;*** POSITION BERECHNEN ***
4380:                  ;
4390:  C526 BD 42 C5             LDA  TAB,X
4390:  C529 85 FA                STA  $FA
4390:  C52B BD 43 C5             LDA  TAB+1,X
4400:  C52E 85 FB                STA  $FB
4410:  C530 68                   PLA
4410:  C531 18                   CLC
4410:  C532 65 FA                ADC  $FA
4410:  C534 85 FA                STA  $FA
4410:  C536 90 02                BCC  POROK1
4410:  C538 E6 FB                INC  $FB
4420:  C53A A0 00      POROK1    LDY  #0
4420:  C53C 68                   PLA
4420:  C53D 91 FA                STA  ($FA),Y
4430:  C53F 4C 01 C7             JMP  CURI
4440:                  ;
4450:  C542 78 04 A0 IAB         .WOR 1144,1184,1224,1264,1304
4460:  C54C 40 05 68             .WOR 1344,1384,1424,1464,1504,1544
4470:  C558 30 06 58             .WOR 1584,1624,1664,1704,1744,1784
4480:  C564 20 07 48             .WOR 1824,1864,1904,1944,1984
4490:                  ;
4500:  C56E 00         FLAG      .BYT 0
4510:                  ;
4520:                  ;*** ZEILE-SPALTE AUSGEBEN ***
4530:                  ;
4540:  C56F A4 58      ZSOUT     LDY  YK
4540:  C571 20 A2 B3             JSR  $B3A2
4540:  C574 20 DD BD             JSR  $BDDD
4550:  C577 A2 1A                LDX  #26
4550:  C579 A0 01                LDY  #1
4550:  C57B 86 D3                STX  $D3
4550:  C57D 84 D6                STY  $D6
4560:  C57F 20 6C E5             JSR  $E56C
4570:  C582 A2 01                LDX  #1
4570:  C584 BD 00 01 ZOUTLAB     LDA  $100,X
4570:  C587 C9 00                CMP  #0
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 15

4570:  C589 F0 07            BEQ  ZOUTEND
4580:  C58B 20 D2 FF         JSR  $FFD2
4580:  C58E E8               INX
4580:  C58F 4C 84 C5         JMP  ZOUTLAB
4590:  C592 E0 03    ZOUTEND CPX  #3
4590:  C594 F0 05            BEQ  SPAL
4600:  C596 A9 20            LDA  #" "
4600:  C598 20 D2 FF         JSR  $FFD2
4610:  C59B A4 57    SPAL    LDY  XK
4610:  C59D 20 A2 B3         JSR  $B3A2
4610:  C5A0 20 DD BD         JSR  $BDDD
4620:  C5A3 A2 24            LDX  #36
4620:  C5A5 A0 01            LDY  #1
4620:  C5A7 86 D3            STX  $D3
4620:  C5A9 84 D6            STY  $D6
4630:  C5AB 20 6C E5         JSR  $E56C
4640:  C5AE A2 01            LDX  #1
4640:  C5B0 BD 00 01 ZOUTLAB2 LDA  $100,X
4640:  C5B3 C9 00            CMP  #0
4640:  C5B5 F0 07            BEQ  ZOUTEND2
4650:  C5B7 20 D2 FF         JSR  $FFD2
4650:  C5BA E8               INX
4650:  C5BB 4C B0 C5         JMP  ZOUTLAB2
4660:  C5BE E0 03    ZOUTEND2 CPX  #3
4660:  C5C0 F0 05            BEQ  ENDZS
4670:  C5C2 A9 20            LDA  #" "
4670:  C5C4 20 D2 FF         JSR  $FFD2
4680:  C5C7 60       ENDZS   RTS
4690:  C5C8 A9 19    SET25   LDA  #25
4690:  C5CA 85 D3            STA  $D3
4690:  C5CC 4C 6C E5         JMP  $E56C
4700:  C5CF A9 12    SET18   LDA  #18
4700:  C5D1 85 D3            STA  $D3
4700:  C5D3 4C 6C E5         JMP  $E56C
4710:  C5D6 A9 1B    SET27   LDA  #27
4710:  C5D8 85 D3            STA  $D3
4710:  C5DA 4C 6C E5         JMP  $E56C
4720:                ;
4730:                ;*** ALLE SPRITES LOESCHEN ***
4740:                ;
4750:  C5DD A9 80    LLAB1   LDA  #128
4750:  C5DF 85 02            STA  2
4750:  C5E1 20 3D C0         JSR  LOESCH
4760:                ;
4770:  C5E4 20 F3 C1 LLAB2   JSR  UEBERN
4770:  C5E7 20 BD C3         JSR  NUMOUT
4780:  C5EA C6 02            DEC  2
4780:  C5EC D0 F6            BNE  LLAB2
4790:  C5EE A9 01            LDA  #1
4790:  C5F0 85 02            STA  2
4790:  C5F2 4C BD C3         JMP  NUMOUT
4800:                ;
4810:                ;**************************************
4820:                ;*** TURN ****************************
4830:                ;**************************************
4840:                ;
4850:  C5F5 20 4A C1 TURN    JSR  REITPIC
4860:  C5F8 A9 36            LDA  #54
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0     PAGE: 16

4860:  C5FA 85 01            STA  1
4870:  C5FC 20 07 C6         JSR  STURN
4880:  C5FF A9 37            LDA  #55
4880:  C601 85 01            STA  1
4890:  C603 20 F3 C1         JSR  UEBERN
4890:  C606 60               RTS
4900:                  ;
4910:  C607 A9 78    STURN   LDA  #<WER1
4910:  C609 A0 04            LDY  #>WER1
4920:  C60B 85 FA            STA  $FA
4920:  C60D 84 FB            STY  $FB
4930:  C60F A9 20            LDA  #<WER2
4930:  C611 A0 A3            LDY  #>WER2
4940:  C613 85 FC            STA  $FC
4940:  C615 84 FD            STY  $FD
4950:                  ;
4960:  C617 A2 15            LDX  #21
4970:                  ;
4980:  C619 A0 18    TLAB1   LDY  #24
4990:  C61B B1 FC    TLAB2   LDA  ($FC),Y
5000:  C61D 91 FA            STA  ($FA),Y
5010:  C61F 88               DEY
5020:  C620 D0 F9            BNE  TLAB2
5030:                  ;
5040:  C622 20 31 C0         JSR  PLUSC40
5050:                  ;
5060:  C625 A5 FC            LDA  $FC
5060:  C627 38               SEC
5060:  C628 E9 28            SBC  #40
5060:  C62A 85 FC            STA  $FC
5060:  C62C B0 02            BCS  TOK1
5070:  C62E C6 FD            DEC  $FD
5080:  C630 18       TOK1    CLC
5080:  C631 CA               DEX
5090:  C632 D0 E5            BNE  TLAB1
5100:  C634 60               RTS
5110:                  ;
5120:                  ;**************************************
5130:                  ;*** ROTATE ***************************
5140:                  ;**************************************
5150:                  ;
5160:  C635 20 4A C1 ROTATE   JSR  REITPIC
5160:  C638 A9 36            LDA  #54
5160:  C63A 85 01            STA  1
5170:  C63C 20 46 C6         JSR  ROT
5180:  C63F A9 37            LDA  #55
5180:  C641 85 01            STA  1
5190:  C643 4C F3 C1         JMP  UEBERN
5200:                  ;
5210:  C646 A9 00    ROT     LDA  #0
5210:  C648 85 FC            STA  $FC
5210:  C64A A9 A0            LDA  #$A0
5210:  C64C 85 FD            STA  $FD
5220:                  ;
5230:  C64E A2 15            LDX  #21
5240:  C650 A9 98    RLAB1   LDA  #<WER1+800
5240:  C652 A0 07            LDY  #>WER1+800
5250:  C654 85 FA            STA  $FA
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 17

5250:  C656 84 FB              STY  $FB
5260:  C658 A0 15              LDY  #21
5270:  C65A B1 FC     RLAB2    LDA  ($FC),Y
5270:  C65C 8D 9A C6           STA  TZW
5270:  C65F 98                 TYA
5270:  C660 48                 PHA
5270:  C661 8A                 TXA
5270:  C662 A8                 TAY
5270:  C663 AD 9A C6           LDA  TZW
5270:  C666 91 FA              STA  ($FA),Y
5270:  C668 68                 PLA
5270:  C669 A8                 TAY
5280:  C66A A5 FA              LDA  $FA
5280:  C66C 38                 SEC
5280:  C66D E9 28              SBC  #40
5280:  C66F 85 FA              STA  $FA
5290:  C671 B0 02              BCS  ROK1
5290:  C673 C6 FB              DEC  $FB
5300:  C675 88         ROK1    DEY
5310:  C676 D0 E2              BNE  RLAB2
5320:                  ;
5330:  C678 20 E2 C0           JSR  PLUSB40
5340:  C67B CA                 DEX
5340:  C67C D0 D2              BNE  RLAB1
5350:                  ;
5360:                  ; 3 ZEILEN LOESCHEN
5370:                  ;
5380:  C67E A2 15              LDX  #21
5390:  C680 A9 78              LDA  #<WER1
5390:  C682 A0 04              LDY  #>WER1
5400:  C684 85 FA              STA  $FA
5400:  C686 84 FB              STY  $FB
5410:  C688 A0 18     DELAB1   LDY  #24
5420:  C68A A9 20     DELAB2   LDA  #32
5420:  C68C 91 FA              STA  ($FA),Y
5430:  C68E 88                 DEY
5430:  C68F C0 15              CPY  #21
5430:  C691 D0 F7              BNE  DELAB2
5440:  C693 20 31 C0           JSR  PLUSC40
5450:  C696 CA                 DEX
5450:  C697 D0 EF              BNE  DELAB1
5460:  C699 60                 RTS
5470:                  ;
5480:  C69A 00         TZW     .BYT 0
5490:                  ;
5500:                  ;*** BLOCK UNTERS ROM KOPIEREN ***
5510:                  ;
5520:  C69B 20 15 C3   KOPBLOCK JSR  UMRECH
5530:  C69E A0 40              LDY  #64
5540:  C6A0 B9 40 03   KOPLAB   LDA  832,Y
5540:  C6A3 91 F8              STA  ($F8),Y
5550:  C6A5 88                 DEY
5550:  C6A6 D0 F8              BNE  KOPLAB
5560:  C6A8 4C AF C2           JMP  KFGF
5570:                  ;
5580:  C6AB 00         FLAG2    .BYT 0
5590:                  ;
5600:                  ;*** TASTERTURTABELLE ***
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 18

5610:                  ;
5620:  C6AC 11         TASTAB   .BYT 17       ; CURSOR DOWN
5630:  C6AD 91                  .BYT 145      ; CURSOR UP
5640:  C6AE 1D                  .BYT 29       ; CURSOR RIGHT
5650:  C6AF 9D                  .BYT 157      ; CURSOR LEFT
5660:  C6B0 93                  .BYT 147      ; CLR EDIT-SPRITE
5670:  C6B1 13                  .BYT 19       ; CURSOR HOME
5680:  C6B2 2B                  .BYT 43       ; PLUS UMSCHALTUNG
5690:  C6B3 2D                  .BYT 45       ; MINUS UMSCHALTUNG
5700:  C6B4 59                  .BYT "Y"      ; SCROLL UP
5710:  C6B5 42                  .BYT "B"      ; SCROLL DOWN
5720:  C6B6 47                  .BYT "G"      ; SCROLL LEFT
5730:  C6B7 48                  .BYT "H"      ; SCROLL RIGHT
5740:  C6B8 53                  .BYT "S"      ; SPIEGELN
5750:  C6B9 52                  .BYT "R"      ; ROTIEREN
5760:  C6BA 18                  .BYT 24       ; BEENDEN
5770:  C6BB 8D                  .BYT 128+13   ; BEENDEN
5780:  C6BC 20                  .BYT 32       ; PUNKT SETZEN
5790:  C6BD A0                  .BYT 160      ; PUNKT LOESCHEN
5800:  C6BE 49                  .BYT "I"      ; INVERTIEREN
5810:  C6BF 46                  .BYT "F"      ; AUSFUELLEN
5820:  C6C0 85                  .BYT 133      ; CCOL+1
5830:  C6C1 89                  .BYT 137      ; CCOL-1
5840:  C6C2 86                  .BYT 134      ; SCOL+1
5850:  C6C3 8A                  .BYT 138      ; SCOL-1
5860:  C6C4 0D                  .BYT 13       ; RETURN
5870:                  ;
5880:  C6C5 F7 C6 FC   WERTAB   .WOR CUDO,CUUP,CURI,CULE
5890:  C6CD 53 C7 0B            .WOR SPRCLR,CUHOME,PLUM,MIUM
5900:  C6D5 68 C0 EE            .WOR SCROLO,SCROLU,SCROLL,SCROLR
5910:  C6DD F5 C5 35            .WOR TURN,ROTATE,END,END,PSET,PER
5920:  C6E9 4A C0 21            .WOR INVERT,FILL
5930:  C6ED 2E C7 39            .WOR CCOLP1,CCOLM1,SCOLP1,SCOLM1
5940:  C6F5 59 C7               .WOR RETURN
5950:                  ;
5960:                  ;
5970:                  ;*********************************
5980:                  ;*** CURSOR DOWN ****************
5990:                  ;*********************************
6000:                  ;
6010:  C6F7 E6 58      CUDO     INC  YK
6020:  C6F9 4C 59 C4            JMP  CUR
6030:                  ;
6040:                  ;*********************************
6050:                  ;*** CURSOR UP ******************
6060:                  ;*********************************
6070:                  ;
6080:  C6FC C6 58      CUUP     DEC  YK
6090:  C6FE 4C 59 C4            JMP  CUR
6100:                  ;
6110:                  ;*********************************
6120:                  ;*** CURSOR RIGHT ***************
6130:                  ;*********************************
6140:                  ;
6150:  C701 E6 57      CURI     INC  XK
6160:  C703 4C 59 C4            JMP  CUR
6170:                  ;
6180:                  ;*********************************
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 19

6190:                  ;*** CURSOR LEFT ****************
6200:                  ;*********************************
6210:                  ;
6220:  C706 C6 57      CULE      DEC  XK
6230:  C708 4C 59 C4              JMP  CUR
6240:                  ;
6250:                  ;*********************************
6260:                  ;*** CURSOR HOME *****************
6270:                  ;*********************************
6280:                  ;
6290:  C70B A9 01      CUHOME    LDA  #1
6300:  C70D 85 57                STA  XK
6310:  C70F 85 58                STA  YK
6320:  C711 4C 59 C4             JMP  CUR
6330:                  ;
6340:                  ;*********************************
6350:                  ;*** END *************************
6360:                  ;*********************************
6370:                  ;
6380:  C714 68        END       PLA
6380:  C715 68                   PLA
6380:  C716 A9 00                LDA  #0
6380:  C718 8D 15 D0             STA  53248+21
6380:  C71B A9 93                LDA  #147
6380:  C71D 20 D2 FF             JSR  $FFD2
6380:  C720 60                   RTS
6390:                  ;
6400:                  ;*********************************
6410:                  ;*** FILL ************************
6420:                  ;*********************************
6430:                  ;
6440:  C721 A9 78      FILL      LDA  #<1144
6450:  C723 A0 04                LDY  #>1144
6460:  C725 85 FA                STA  $FA
6460:  C727 84 FB                STY  $FB
6460:  C729 A9 A0                LDA  #160
6460:  C72B 4C 1F C0             JMP  SKIP1
6470:                  ;
6480:                  ;*********************************
6490:                  ;*** CCOL PLUS 1 *****************
6500:                  ;*********************************
6510:                  ;
6520:  C72E E6 59      CCOLP1    INC  CURCOL
6530:  C730 20 9C C4   CCOLP1B   JSR  CUROUT
6540:  C733 A5 59                LDA  CURCOL
6550:  C735 8D 29 D0             STA  53248+41
6560:  C738 60                   RTS
6570:                  ;
6580:                  ;*********************************
6590:                  ;*** CCOL MINUS 1 ****************
6600:                  ;*********************************
6610:                  ;
6620:  C739 C6 59      CCOLM1    DEC  CURCOL
6620:  C73B 4C 30 C7             JMP  CCOLP1B
6630:                  ;
6640:                  ;*********************************
6650:                  ;*** SCOL PLUS 1 *****************
6660:                  ;*********************************
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0     PAGE: 20

6670:                  ;
6680:  C73E E6 F7      SCOLP1    INC  247
6690:  C740 20 D0 C4   SCOLP1B   JSR  SCOLOUT
6700:  C743 A5 F7                LDA  247
6710:  C745 8D 27 D0             STA  53248+39
6710:  C748 8D 28 D0             STA  53248+40
6720:  C74B 4C 15 C0             JMP  SCOLOR
6730:                  ;
6740:                  ;*********************************
6750:                  ;*** SCOL MINUS 1 ***************
6760:                  ;*********************************
6770:                  ;
6780:  C74E C6 F7      SCOLM1    DEC  247
6780:  C750 4C 40 C7             JMP  SCOLP1B
6790:                  ;
6800:                  ;*********************************
6810:                  ;*** CLR SPRITE *****************
6820:                  ;*********************************
6830:                  ;
6840:  C753 20 3D C0   SPRCLR    JSR  LOESCH
6840:  C756 4C 0B C7             JMP  CUHOME
6850:                  ;
6860:                  ;*********************************
6870:                  ;*** RETURN *********************
6880:                  ;*********************************
6890:                  ;
6900:  C759 E6 58      RETURN    INC  YK
6900:  C75B A9 01                LDA  #1
6900:  C75D 85 57                STA  XK
6910:  C75F 4C 59 C4             JMP  CUR
6920:                  ;
6930:                  ;*********************************
6940:                  ;*** EINSPRUNG NACH RUECKKEHR ****
6950:                  ;*** VON EINER BASICROUTINE   ****
6960:                  ;*********************************
6970:                  ;
6980:  C762 A9 07      BASI      LDA  #7
6980:  C764 8D 15 D0             STA  53248+21
6990:  C767 20 AF C2             JSR  KFGF
6990:  C76A 20 59 C4             JSR  CUR
7000:  C76D 20 9C C4             JSR  CUROUT
7000:  C770 20 D0 C4             JSR  SCOLOUT
7010:  C773 4C BD C3             JMP  NUMOUT
7020:                  ;
7030:                  ;*********************************
7040:                  ;*** UNTERROUTINE ZUR RETTUNG  ***
7050:                  ;*** DES GESAMMTEN BILDSCHIRMS ***
7060:                  ;*********************************
7070:                  ;
7080:  C776 78         GPARRS    SEI
7090:  C777 A9 30                LDA  #48
7090:  C779 85 01                STA  1
7100:  C77B A9 00                LDA  #<1024
7100:  C77D A0 04                LDY  #>1024
7110:  C77F 85 FA                STA  $FA
7110:  C781 84 FB                STY  $FB
7120:  C783 A9 00                LDA  #<53248
7120:  C785 A0 D0                LDY  #>53248
```

```
SPRITE-EDITOR V1.0     PROFI-ASS 64 V2.0    PAGE: 21

7130:  C787 85 FC            STA  $FC
7130:  C789 84 FD            STY  $FD
7140:  C78B A0 00            LDY  #0
7140:  C78D A2 03            LDX  #3
7140:  C78F 60               RTS
7150:                ;
7160:  C790 20 76 C7 RETS    JSR  GPARRS
7170:                ;
7180:  C793 B1 FA    RSLAB1  LDA  ($FA),Y
7180:  C795 91 FC            STA  ($FC),Y
7190:  C797 C8               INY
7190:  C798 D0 F9            BNE  RSLAB1
7190:  C79A E6 FB            INC  $FB
7190:  C79C E6 FD            INC  $FD
7190:  C79E CA               DEX
7190:  C79F 10 F2            BPL  RSLAB1
7200:  C7A1 A9 37    RSEND   LDA  #55
7200:  C7A3 85 01            STA  1
7200:  C7A5 58               CLI
7200:  C7A6 60               RTS
7210:                ;
7220:  C7A7 20 76 C7 GETSC   JSR  GPARRS
7230:                ;
7240:  C7AA B1 FC    RSLAB2  LDA  ($FC),Y
7240:  C7AC 91 FA            STA  ($FA),Y
7250:  C7AE C8               INY
7250:  C7AF D0 F9            BNE  RSLAB2
7250:  C7B1 E6 FB            INC  $FB
7250:  C7B3 E6 FD            INC  $FD
7250:  C7B5 CA               DEX
7250:  C7B6 10 F2            BPL  RSLAB2
7260:  C7B8 4C A1 C7         JMP  RSEND
7270:                ;
7280:                ;**********************************
7290:                ;*** COLOR FUER DEN GESAMMTEN BS *
7300:                ;**********************************
7310:                ;
7320:  C7BB A9 00    CFGBS   LDA  #<55296
7320:  C7BD A0 D8            LDY  #>55296
7330:  C7BF 85 FA            STA  $FA
7330:  C7C1 84 FB            STY  $FB
7340:  C7C3 A2 03            LDX  #3
7340:  C7C5 AD 86 02         LDA  646
7340:  C7C8 A0 00            LDY  #0
7350:  C7CA 91 FA    CFGBSLAB STA  ($FA),Y
7350:  C7CC C8               INY
7360:  C7CD D0 FB            BNE  CFGBSLAB
7370:  C7CF E6 FB            INC  $FB
7380:  C7D1 CA               DEX
7380:  C7D2 10 F6            BPL  CFGBSLAB
7380:  C7D4 60               RTS
]C000-C7D5
NO ERRORS
```

```
SPRITE-EDITOR V1.0      PROFI-ASS 64 V2.0     PAGE: 22

SYMBOLTABLE:
CFGBSLAB  C7CA   CFGBS    C7BB   RSLAB2   C7AA   GETSC    C7A7
RSEND     C7A1   RSLAB1   C793   RETS     C790   GPARRS   C776
BASI      C762   RETURN   C759   SPRCLR   C753   SCOLM1   C74E
SCOLP1B   C740   SCOLP1   C73E   CCOLM1   C739   CCOLP1B  C730
CCOLP1    C72E   FILL     C721   END      C714   CUHOME   C70B
CULE      C706   CURI     C701   CUUP     C6FC   CUDO     C6F7
WERTAB    C6C5   TASTAB   C6AC   FLAG2    C6AB   KOPLAB   C6A0
KOPBLOCK  C69B   TZW      C69A   DELAB2   C68A   DELAB1   C688
ROK1      C675   RLAB2    C65A   RLAB1    C650   ROT      C646
ROTATE    C635   TOK1     C630   TLAB2    C61B   TLAB1    C619
STURN     C607   TURN     C5F5   LLAB2    C5E4   LLAB1    C5DD
SET27     C5D6   SET18    C5CF   SET25    C5C8   ENDZS    C5C7
ZOUTEND2  C5BE   ZOUTLAB2 C5B0   SPAL     C59B   ZOUTEND  C592
ZOUTLAB   C584   ZSOUT    C56F   FLAG     C56E   TAB      C542
POROK1    C53A   PSET     C519   PER      C516   T3       C50E
T2        C506   CO2END   C503   COLAB2   C4F5   COOK1    C4DA
SCOLOUT   C4D0   COEND    C4CD   COLAB    C4C1   OOK1     C4A6
CUROUT    C49C   ABOK4    C481   ABOK3    C479   ABOK2    C46F
ABOK1     C465   CUR      C459   CSLAB    C43D   LOLAB    C431
INCURS    C42E   RUECK    C428   FOUND    C412   TASTLAB  C405
TASTER    C3FE   T1       C3F6   NLAB2    C3ED   ENDD     C3EB
NLAB1     C3D9   NUMOUT   C3BD   INIT     C365   MIOK     C35F
MIUM      C357   PLOK     C351   PLUM     C344   UMROK    C336
UMRLAB    C326   UMRECH   C315   WOK2     C2EB   WLAB2    C2E0
WLAB1     C2CF   WLAB3    C2C8   UMWNDL   C2BA   KFGF     C2AF
FERTIG    C2AE   CSRLAB   C29C   CLRROM   C288   GLAB2    C277
GLAB1     C275   GETS     C273   UOK1     C21E   ULAB2    C20E
ULAB3     C205   UEBERN   C1F3   MUSTER   C1EB   SPRITE   0340
ZEILE     C1EA   BITMUS   C1E9   DREI     C1E8   SCOUNT   C1E7
SRLAB2    C1D1   SRLAB1   C1C9   RSCROL   C1C7   SCROLR   C1BB
SLLAB2    C1A3   SLLAB1   C19B   LSCROL   C199   SCROLL   C18D
GETWER6   C17C   GETWER5  C16B   RETLAB2  C15A   RETLAB1  C158
RET1      C156   RETPIC   C14A   OZLAB1   C128   OZEILE   C126
SULAB2    C116   SULAB1   C114   SCU      C112   SCUU     C0FA
SCROLU    C0EE   PBOK     C0EB   PLUSB40  C0E2   BUF      C0C8
GETWER1   C0B6   GETWER2  C0A5   SLAB4    C09C   SLAB3    C08A
SLAB2     C088   SLAB1    C078   SCROLO   C068   LAB4     C056
LAB3      C054   INVERT   C04A   LOESCH   C03D   OK       C03A
PLUSC40   C031   LAB2     C023   LAB1     C021   SKIP1    C01F
SCOLOR    C015   WER2     A320   WER1     0478   GET      FFE4
GRUNDY    0042   GRUNDX   0018   CURCOL   0059   YK       0058
XK        0057   ANF      C000
166 SYMBOLS DEFINED
```

Register

A
Animation 3.1
Anti-Error-Routine 1.9
Anwender 4.1
Array-Element 1.27
Arrays 1.1
ASCII 7.3
Assembler 4.4
Attack 3.3
Autoline 2.16
Autostart 8.12

B
BAM 5.1
BASIC 1.8
BASIC-Hardcopy 8.10
BCD-Format 7.2
Befehlserweiterung 3.3, 6.1, 6.2, 6.6
Berechnete Zeilennummern 1.25
Betriebssystem 4.5, 6.1, 6.6, 7.1
Bildschirm (Ein/Aus) 3.1.3, 8.17
Bildschirmmasken-Generator 1.23
Bildschirmzonen 2.7
Block 5.1
Border-Farben 2.2.1
Bubble-Sort 1.27
Buffer 5.15, 6.5
Busy-Leitung 7.3
Byte 7.2

C
Cassette 5.15
Cassetten-Buffer 4.6, 5.15, 6.5
Centronics 7.3
Centronics-Schnittstelle 7.4
Character-Generator 2.5, 2.51
CIA 7.1

CIA Uhr 1.11
Clock-Register 1.11
CMD-Modus 7.3
Codewort 5.1
Collection 6.2
Commodore-Taste 8.32
Control-Port 4.1
Control-Taste 4.6, 8.32
CTRL 4.5
Cursor 4.1
Cursor beschleunigen 8.20
Cursor positionieren 1.18, 1.19
Cursor-Blinkfrequenz 1.10
Cursor-Steuerung 4.2

D
DATA-Zeilen 8.7
Datafile 5.4
Datasette-Sound 3.1.1
Datasette-Steuerung 8.18
Datazeilengenerator 1.2.2
Daten-Felder 6.3
Datenbit 7.2
Datenbyte 7.3
Datenpuffer 5.1
Datenrichtung 7.2
Datenübertragung 7.4
Decay 3.3
DIM-Anweisung 1.1, 1.3
Directory 5.6, 5.9-5.14, 8.1
Dreieckswellenform 3.3
Drucker 7.3
Dummy 4.3

E
Echo 3.1.5
Echtzeituhr 7.2
Editieren 4.2
Eingaberoutine 1.20
Einzelpunktgrafik 7.3

Empfänger 7.4
Envelope-Befehl 3.3
Ergonomie 4.1
Error-Modus 1.9
Error-Seite 1.9

F
FAC 6.1
Fakutätsfunktionen 1.6
Fehlermeldung 4.4
Fehlermeldungen erzwingen 3.1.1, 8.19
Fenstertechnik 4.3
Feuerknopf 4.3
Filenummer 5.2
Filter 3.1.2
Flasheffekte 2.3
Flasher 1.12
Fließkomma 6.1, 6.2
Fließkommazahleneingabe 1.20
Floatvariable 1.14
Formula to complex 8.5
FRE 6.2
Freies RAM 8.1
Frequenzbereiche 3.2
Funktionen 1.8
Funktionstasten 4.6
Funktionstastenabfrage 4.6
Funktionstastensplitting 8.32

G
Garbage Collection 6.2
GEM 4.3
GEM-Simulation 4.3
Get#-Beschleunigung 8.27
Grad 1.8.4
Grafik 3-D 2.4.3
Grafikanimation 2.2

H
Halbton 3.3
Handshake-Leitung 7.3, 7.4
Hardcopy 2.4.2, 5.2, 8.1
Hardwareschutz 5.1
High-Byte 7.2
hochauflösende Grafik 2.4
Hüllkurve 3.3.

I
Illegal-Voice-Number-Error 3.3
INPUT-Befehl 1.20, 8.9
Input-Routine in Assembler 1.20
INSTR 6.4
Integer-Format 6.1
Integer-Funktion 8.29
Integer-Variable 1.14
Interface 4.1
Interrupt 4.2
Interrupt-Zeiger 4.2

J
JIFFI-Clock 1.10
Joystick 4.1, 4.3, 5.15

K
Kaltstart 8.31
Kassette 7.4
Kassetten-Puffer 1.26, 4.6
Klangverbesserung 3.2
Kopier-Programm 5.1
Kopier-Schutz 5.1, 8.22

L
Ladefehler 5.15
Laden von Maschinenspracheprogrammem 8.16
Lautstärke 3.1.1
Leitungslänge 7.4
LINE-DELETE 1.15

List-Schutz 8.15
List-Vektor 8.15
Logarithmus 1.8.2
Low-Byte 7.2

M
Makros 1.8
Maskenbit 7.2
Matrix 4.5
Matrix-Tabelle 4.5
Maus 4.1
Maus-Simulation 4.1, 4.2
Menü-Struktur 4.3
Menütechnik 4.1
Merge per Hand 1.26
Micro-Cursor 2.5.2
Minuten 1.10
Modifizierter PRINT-Befehl 1.19
Modulo 1.8.3

N
NMI 4.5
NMI-Vektor 8.2

O
Oktave 3.3
Out of memoy error 1.17

P
Paddle 4.1
Paperfarbe 2.2.3
Parameter 4.3
Personalcomputer 4.1
Port 7.1
Print Fre () 1.17
Print-Befehl 1.5
Programm zurückholen 8.4
Programm-Absturz 1.9

Programm-Generator 1.21
Programme als Sequ.-/User-Files 8.2.2
Programmierung der TOD-Clock 1.11
Pull-Down-Menü 4.3

R
Radians 1.8.4
Rasterzeilen-Interrupt 2.7
Rauschen 3.1.2
Read 7.2
Ready Modifikation 8.24
Ready-Meldung unterdrücken 8.11
Register 7.2
Release 3.3
Rename 5.14
RENEW 8.4
Reset 5.1, 5.7, 5.10, 8.31
Reset blockieren 8.13
Resonanz-Filter 3.1.2
Restore 4.5
Restore-Taste blockieren 8.2
REVERSE-Befehl 1.13
Rhythmus 3.1.3
Ring-Modulation 3.3
ROM in's RAM kopieren 8.24
Routine 4.2, 4.3
RUN/STOP-RESTORE 8.2
Rundung 1.8.7

S
Schnittstelle 7.1
Schreibschutz 5.3, 5.4, 5.5, 5.11
Schrittweite 1.16
Scratch 5.10
Screen ON/OFF 1.4
Scroll-Down 8.23
Sektor 5.1
Sekundär-Adresse 7.3
Sekunden 1.10
SEQ-File 5.6

SEQU.-Files 8.22
Sequentiell 5.6
Shift-Taste 4.5, 8.32
SID 3.3, 8.21
Simulation 4.2, 4.3
Sound 3.1
Sound-Animation 3.1.2-3.1.5
Sound-Dämmung 3.1.4
Sounderweiterung 3.3
Soundstimmen 3.3
Space-Tracer 1.7
Spalten 4.5
Speicherende ermitteln 1.21
Speicherkonfiguration 4.5
Spielautomat 4.1
Spinball 4.1
Sprite 2.5.2, 2.6, 2.6.1, 4.3
Sprite-Editor 2.6.2, 2.6.3
Sprung-Vektor 6.6
Spur 5.1
Stack für Strings 8.3
Stack overflow 1.8.7, 1.17
Stack-Manipulation 8.12, 8.14
Statement-Flasher 1.12
Statuszeilen in Programmen 1.18
Stereo 3.2
Steuercodes 5.13
Steuerung der Datasette 8.18
Stoppuhr 1.10
String-Eingabe 1.20
String-Stack 8.3
Strobe 7.2, 7.3
Struktuiertes Programmieren 1.6, 8.8
Stunden 1.11
Sustain 3.3
Synthesizer 3.2
SYS-Befehl 1.19
SYS-Befehlserweiterung 3.3
System-Takt 7.2

T
Tastatur 4.6
Tastaturabfrage 4.5
Tastaturbelegung 4.5
Tastaturpips 1.20
Tastennummer 4.5
Tatatureingabepuffer 1.15, 1.22, 4.6
TEC-Modus 2.3
TEC-Reference-Liste 2.3.1
Textgrafik 1.28
Textverarbeitung 8.3
TI 1.10
TI$ 1.10
TIME$ 1.10
Time-Verzögerung 8.2.3
Timer 7.2
TOD-Clock 1.11
Token 6.6
Ton 3.1
Tonsignal 3.2
Trigger 7.2

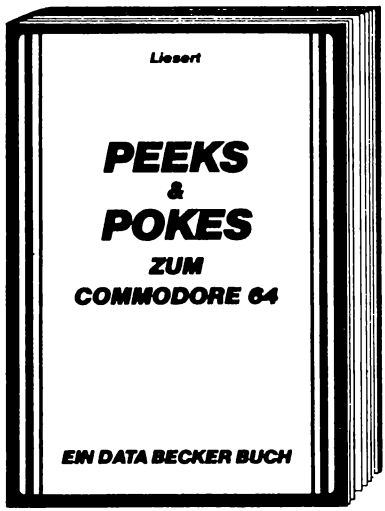
U
Uhren 1.11
unerlaubte Variablen 8.30
User-File 8.22
Userport 7.1
USR 5.6

V
Variablen Tracer 1.17
Variablen-Dump 1.14
Vektor-NMI 8.2
Vektor auf erste BASIC-Zeile 8.4
Vektor auf letzten String 8.5
Vektor auf Reset 8.13
Vektor auf Stoproutine 8.2
Vektor für LIST 8.15
Verkürzen der Floppy-Zugriffszeit 8.6
Verzweigungs-Adresse 4.3

VIC 2.7, 8.21
Videocontroller 2.7
Videospiel-Automat 4.1
Volume-Befehl 3.3

W
Warmstart 6.6
Wave-Befehl 3.3
Wellenform 3.3
Winkelcodierer 4.1
Write 7.2

Z
Zahleneingabe 1.20
Zeichensatz verändern 2.5.2
Zeichensatz verschieben 2.5.1
Zeichensatzmodifikation 2.5
Zeiger auf erste BASIC-Zeile 8.4
Zeiger auf letzten String 8.5
Zeiger für LIST 8.15
Zeiger-Reset 8.13
Zeilen 4.5
Zeilennummern 1.16
Zeitverzögerungsschleifen 8.23
Zentrieren 1.8.1
Zero-Page 1.18, 8.2
Zufallsroutine 4.3



Schauen Sie ins Innere Ihres Rechners! Leichtverständlich wird in diesem Buch der Umgang mit PEEK- und POKE-Befehlen erklärt. Außerdem Grundlegendes zum Aufbau des C-64: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks, Charakter-Generator, Sprite-Register und vieles mehr. Mit einer ersten Einführung in die Maschinensprache und etlichen Beispielprogrammen.

Liesert
Peeks & Pokes zum Commodore 64
177 Seiten, DM 29,–
ISBN 3-89011-032-0

Ein Bestseller, der umfassend in die Maschinensprache einführt! Sie lernen Aufbau und Arbeitsweise des 6510 Prozessors kennen und erfahren Wichtiges über Eingabe und Start von Maschinenprogrammen. Assembler, Disassembler und ein Einzelschrittsimulator sind als Programme im Buch enthalten. Viele ausführlich beschriebene Beispielprogramme und Routinen machen Ihnen den Einstieg leicht!

Englisch
Das Maschinensprachebuch zum Commodore 64 & C128
201 Seiten, DM 39,–
ISBN 3-89011-008-8

Auch der zweite Band weckt Experimentierfreude. Mit umfangreichen Kapiteln über Softwareschutz, Pointer und deren Manipulation, Interrupthandling, ROM ins RAM kopieren, Softwareschutz, Arbeiten mit 2 Bildschirmen, modifiziertes Input und vielen anderen nützlichen Befehlserweiterungen und Routinen. Für Leute, die gerne programmieren!

Hornig/Trapp/Weltner
64 Tips & Tricks, Band 2
259 Seiten, DM 39,–
ISBN 3-89011-065-7

Die Herausforderung für jeden ernsthaften C-64 Anwender! Alles über Technik, Betriebssystem und Programmierung des Commodore 64: kommentiertes ROM-Listing, Speicherbelegungspläne, SID- und VIC-Chip, Ein-/Ausgabesteuerung, Timer und Echtzeituhr, BASIC-Interpreter, mathematische Routinen, IEC-Bus, RS-232, Zeropage und Originalschaltpläne. Das Standardwerk zur Hardware des C-64!

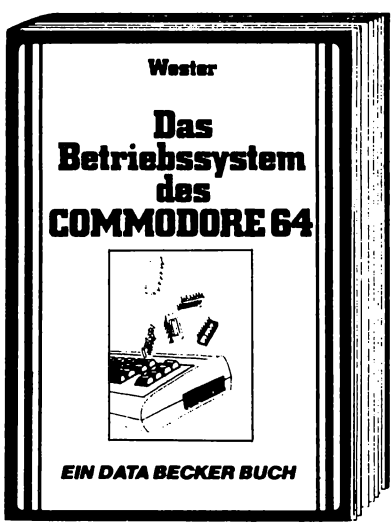
Angerhausen/Brückmann/
Englisch/Gerits
64 Intern
352 Seiten, 2 Schaltpläne,
DM 69,–
ISBN 3-89011-000-2

BASIC leichtgemacht! Das bietet dieser komplette Programmierkurs in der preiswerten Buchform. Mit ihm lernen Sie von Grund auf das Beherrschen der einzelnen Befehle und ihre Anwendungen. Vieles über die Grundlagen des Programmierens, über BIT, BYTE und ASCII-Code, Programmablaufpläne, Unterprogramme und Menuetechniken. Alles was Sie für Ihre zukünftigen Programmiererfolge benötigen.

Kampow
Das BASIC-Trainingsbuch zum Commodore 64
308 Seiten, DM 39,–
ISBN 3-89011-023-1

Maschinensprache für Profis! Zahlendarstellung, Interruptprogrammierung, Betriebssystem- und BASIC-Erweiterungen sind die Themen dieses Buches. Dazu viele Assemblerprogramme: Sortieren von Zahlenfeldern, Cursorveränderungen, 2 Bildschirme, User-Port, Speicherplatzberechnung, 16 Sprites, Echtzeituhr mit Wecker, Interruptgesteuerte BASIC-Unterprogramme u.v.m. Auch für den C-128!

Englisch
Das Maschinensprachebuch für Fortgeschrittene zum Commodore 64 & C-128
207 Seiten, DM 39,–
ISBN 3-89011-022-3

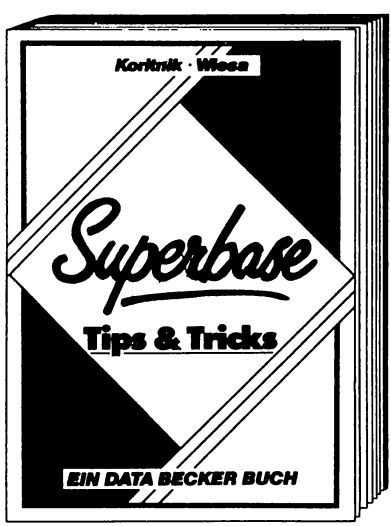


79 (!) Routinen des Betriebssystems enthält dieses Buch. Z.B.: Eingabe einer Zeile per Tastatur, String ausgeben, Ausgabe eines ASCII-Zeichens, beliebigen Ausdruck holen, Multiplikation/Division und Cursor setzen/holen. Startadresse, Einsprungbedingungen, Akku, Register und Flags werden jeweils beschrieben. Ein unverzichtbares Hilfsmittel für jeden Maschinensprachepro-grammierer!

Wester
Das Betriebssystem des Commodore 64
177 Seiten, DM 29,–
ISBN 3-89011-136-X

Was! Sie wissen nicht, was DFÜ ist? Dann müssen Sie dieses Buch lesen! Es führt Sie umfassend in die Welt der Datenübertragung ein: Grundbegriffe, Soft- und Hardware für die eigene Mailbox, Akustikkoppler zum Selbstbauen, notwendige Schnittstellen und Kosten der DFÜ. Hacker sollten zum Schluß die Kapitel über rechtliche Bestimmungen, Datenschutz und Copyright lesen!

Severin/Schulwitz
DFÜ für Jedermann, Datenfernübertragung mit dem C-64 & C-128
über 250 Seiten, DM 39,–
ISBN 3-89011-141-6



Neben einer Einweisung in SUPERBASE und seine Befehle enthält das Buch eine riesige Anzahl Tips & Tricks: Datensätze pro Diskette, gestaffelte Suche, nachträgliches Ändern des Schlüsselfeldes, Sortieren nach Zahlen in Textfeldern, Benutzung von 2 Laufwerken, Übernahme eines Formates in eine andere Datei, Datei nachträglich splitten. Dieses Buch zeigt alle Möglichkeiten des SUPERBASE!

Koritnik/Wiesa
Superbase Tips & Tricks
266 Seiten, DM 49,–
ISBN 3-89011-047-9

Ein faszinierender Führer in die Welt der Abenteuerspiele. Hier läßt sich ein arrivierter Autor in die Karten gucken. Er zeigt, wie Adventures funktionieren und wie man sie programmiert. Der Clou des Buches ist neben fertigen Adventures zum Abtippen ein kompletter ADVENTURE-GENERATOR, mit dem das Selbsterstellen zum Kinderspiel wird. Achtung: dieses Buch macht süchtig!

Walkowiak
Adventures – und wie man sie programmiert
225 Seiten, DM 39,–
ISBN 3-89011-043-6

Empfängerabschnitt

DM	Pf
19	80

für Postgirokonto Nr.
789 - 436

Absender (mit Postleitzahl)

Verwendungszweck
Diskette z. Buch
376 001

Zahlkarte

(Mit Schreibmaschine, Tinte oder Kugelschreiber deutlich ausfüllen)

DM	Pf	für Postgirokonto Nr.
19	80	789 - 436

Absender:

DM	Pf	(DM-Betrag in Buchstaben wiederholen)
19	80	Neunzehn

für
DATA BECKER GmbH
Merowingerstr. 30

in
4000 Düsseldorf

Postgirokonto Nr.
789 - 436

Postgiroamt
Essen

Postvermerk

Für Vermerke des Absenders

Einlieferungsschein

– Bitte sorgfältig aufbewahren –

DM	Pf
19	80

für
DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Postgirokonto Nr.
789 - 436

Postvermerk

LADEN, STARTEN - KLAR!

Für alle diejenigen, die sich fleißiges Abtippen ersparen und trotzdem alle Programme nutzen wollen, gibt es einen Ausweg:

Mit der nebenstehenden Post-Zahlkarte einfach die Diskette zum Buch bestellen!

Diskette zum Buch „64 Tips & Tricks“ DM 19,80

Diskette zum Buch
„64 Tips & Tricks“
376 001

Für Mitteilungen an den Empfänger

Bedienen Sie sich der Vorteile
eines eigenen Postgirokontos!
Auskunft hierüber erteilt jedes Postamt

Feld
für
postdienstliche
Zwecke

Einlieferungsschein

(nicht zu Mitteilungen an den Empfänger benutzen)

Gebühr für die Zahlkarte
(wird bei der Einlieferung bar erhoben)

bis 10 DM . . . . . 90 Pf
über 10 DM (unbeschränkt) 1,50 DM

DAS STEHT DRIN:

64 Tips & Tricks Band 1, das auflagenstärkste deutsche Computerbuch, bringt in einer komplett überarbeiteten und stark erweiterten Neuauflage alle Tips & Tricks auf einen Blick. Sparen Sie sich das lästige Blättern und Suchen in anderen Büchern und Zeitschriften – mit dem Original können Sie ihre Zeit sofort zum Programmieren verwenden! Jetzt noch mehr Fakten: BASIC-Programmierung effektiver und besser, Grafik für Fortgeschrittene, Soundprogrammierung, die Benutzerschnittstellen, die Peripherie, Befehlserweiterungen, Schnittstellen und ein ganzes Kapitel mit Kurztips.

Aus dem Inhalt:

— Die 3 Uhren des C-64
— Errechnen beliebiger Formeln
— Bubblesort in Assembler
— Grafikanimation
— Hochauflösende 3-D-Grafik
— Synthesizer in Stereo
— Maussimulation mit dem Joystick
— GEM-Simulation auf dem C-64
— Kopierschutz
— User-Port
— Datenübertragung zu anderen Rechnern
— Directory ohne Programmverlust

UND GESCHRIEBEN HAT DIESES BUCH:

Das bewährte DATA-BECKER-Autorenteam mit Lothar Englisch, Klaus Gerits, Olaf Hartwig, Klaus Löffelmann und Frank Thrun. Alle sind nicht nur begeisterte Programmierer, die ihren 64er in- und auswendig kennen, sondern auch bekannte Autoren vieler weiterer Bücher.

ISBN 3-89011-001-1

This was brought to you

from the archives of

http://retro-commodore.eu